Natural Deduction
Sharing by Presentation

Een wetenschappelijke proeve op het gebied
van de Natuurwetenschappen, Wiskunde en Informatica

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus
prof. mr. S.C.J.J. Kortmann,
volgens besluit van het College van Decanen
in het openbaar te verdedigen op
woensdag 20 juni 2007
om 10.30 uur precies
door

Iris Loeb

geboren op 27 maart 1980 te Wageningen

Promotor:

Prof. dr. J.H. Geuvers

Manuscriptcommissie:

Prof. dr. A. Asperti	(Università di Bologna)
Prof. dr. H.P. Barendregt	
Dr. V. van Oostrom	(Universiteit Utrecht)

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

This research was partially supported by the European Project IST-2001-33562 MoWGLI and partially supported by the Spinoza research grant of Prof. dr. H.P. Barendregt.

ISBN 978-90-6464-136-7

Printed by Ponsen & Looijen b.v., Wageningen

Typeset by LaTeX2e

IPA Dissertation Series 2007-06

Acknowledgments

First of all I would like to thank my promotor Herman Geuvers, not only for his scientific guidance, but also for showing me how to work cheerfully. I hope he has enjoyed our collaboration at least as much as I have.

I thank the reading committee, consisting of Andrea Asperti, Henk Barendregt, and Vincent van Oostrom for their careful reading and useful comments. All three of them have also been involved in earlier stages of my research, for which I am very grateful.

The first two years of this research were carried out within the European project MoWGLI. I am indebted to its members. Especially I would like to thank Lionel Mamane, Luca Padovani, Claudio Sacerdoti Coen, and Stefano Zacchiroli.

I thank my roommates Wil Dekkers, Luís Cruz-Filipe, Jasper Stein, and Milad Niqui for their companionship.

Dan Synek never got tired of showing me how I could solve my computer problems by myself. I am very grateful for that.

Freek Wiedijk took care of the realisation of the cover of the thesis, and Maxim Hendriks changed the thesis to the requested size. I would like to thank them for their spontaneous technical help.

I thank the team of Preuves, Programmes et Systèmes of the Université Paris VII– Denis Diderot for being my colleagues during the three months that I visited Paris. Especially I thank Delia Kesner and Stéphane Lengrand for our interesting discussions.

I would also like to thank Ian Mackie, Jan Willem Klop, Femke van Raamsdonk, Bas Spitters, Lorenzo Tortora de Falco, and Freek Wiedijk for showing interest in my research, and for the discussions we have had.

I thank everyone of the Foundations group for creating a nice working environment.

The astrophysicists Roy Smits, Martin van den Akker, and Else van den Besselaar (in order of appearance) have been terrific friends. I hope that I may also count on their friendship in the future.

I thank my family for their support. I am very happy to have Roos and Rahele as my “paranimfen”.

Contents

Introduction

Proof theory can roughly be divided into two categories: Structural proof theory on the one side, and what we will call provability theory on the other side.

Structural proof theory studies formal derivations in a mathematical way. Derivations are seen as geometrical structures. Transformations of a derivation, like removing irrelevant parts and simplifying the argumentation, add a dimension that makes this field interesting.

Provability theory, on the other hand, considers mathematics from a logical perspective. A typical question is: Can this mathematical theorem be proven within this formal system?

Clearly, the above division is not clear-cut. When we know, for example, which geometrical structure a proof may take, it might become easier to search for a proof of a certain theorem. For this thesis, however, it describes nicely the subjects of the two parts: Part I can be placed within structural proof theory, and Part II lies within the field of (applied) provability theory.

In Part I we study a fragment of natural deduction to which we have added the notion of *sharing* of subresults. It is common practice in mathematics to repeatedly use an obtained result. Still, in Gentzen's system for natural deduction [27] this is not taken into account.

Derivations with sharing of subresults have in general a more complicated structure than derivations without sharing. The transformations on these derivations therefore need to be more subtle. This can easily be understood: A subresult might be unnecessary general for one of its uses, at the same time it might not be directly simplifiable because its other uses need the full broadness.

Chapter 1 gives a historical overview of natural deduction and its presentation. We sketch the development of the theory. We also raise an objection to already existing formalisms that allow sharing. By explaining the position of our formalism of natural deduction with sharing, the so-called *deduction graphs*, with respect to this evolution, we motivate our research.

Chapter 2 introduces deduction graphs for minimal proposition logic formally. For this we need a little of the theory of orderings and a little graph theory. We show how other well-known natural deduction formalisms can be embedded in deduction graphs.

Chapter 3 studies the transformations on deduction graphs that we need to describe the elimination of *cuts*.

Chapter 4 treats simply typed λ-calculus and a context calculus with let-bindings, and relates deduction graphs and its transformations to these formalisms. From this we obtain *strong normalisation* for the process of cut-elimination. That is: Every deduction graph can be transformed into a cut-free deduction graph in finitely many transformation steps. It turns out that deduction graphs can be fully described in the context calculus with let-bindings.

Chapter 5 starts with a brief overview of multiplicative exponential linear logic (MELL) and the associated graphical presentation of proof nets [36]. At first glance, deduction graphs resemble proof nets. After changing our attention to *deduction graphs with explicit sharing*, we make this similarity precise. This agreement is not a priori obvious, as proof nets of MELL and deduction graphs originate from different ideas.

Chapter 6 then shows *confluence* for deduction graphs with explicit sharing. That means that the order in which cuts are eliminated has no impact on the end-result. This is obtained by brute force: The proof consists of an exhaustive case-analysis.

Chapter 7 finally adds universal quantification to the deduction graph formalism.

The Chapters 2–4 are a somewhat extended version of [28]; Chapter 5 is an extended version of [29]; [30] is a reduced version of Chapter 7.

Part II is about computer formalised mathematics.

Computers are good bookkeepers. Doing a mathematical proof in such a way that a computer can verify it, may thus lead to a high credibility of the correctness of the result. At the same time, using computer formalised mathematics brings about some problems.

One of the problems is the presentation of the formalised mathematics. In order for a computer to be able to verify a proof, it needs all details of the definitions and of the argumentation. For a human reader, a formal proof seems therefore often too verbose, and thus hard to understand.

Chapter 8 addresses the problem of presentation by a case-study. We replace definitions and theorems of an interactive algebra course by their computer-formalised counterparts. The purpose of the document, education, stresses the importance of readability and clarity. We obtain some degree of readability by letting the formal mathematics undergo a complicated transformation procedure.

This case-study also touches on some related areas: Using libraries of formal mathematics and putting formalised mathematics on the world wide web.

The papers [2] and [3] report also on this research.

The title of this thesis, “Natural Deduction: Sharing by Presentation”, applies to both parts.

Both Part I and Part II have a clear relation to natural deduction. Natural deduction is the subject of Part I, whereas it is used in Part II to formalise mathematics.

In Part I we study the *sharing* of subproofs. In order to do this, we introduce a new *presentation* for natural deduction derivations, because the existing presentations either do not allow sharing, or do not fulfill some of our additional wishes.

To *share* computer-formalised mathematics with others, we supply a reasonable way to present it in Part II. It seems that a good *presentation* is one of the most important issues for the use of computer-formalised mathematics in daily mathematical practice.

Part I

Deduction Graphs

Chapter 1

Historical Overview

1.1 Introduction

The thirties of the last century heralded the start of natural deduction. Two systems defining a formal system for natural deduction for proposition and predicate logic appeared at almost the same time: One by Jaśkowski [45][1] and one by Gentzen [27][2].

Stanisław Jaśkowski was a Polish mathematician, whose 1934 paper "On the Rules of Suppositions in Formal Logic" was an answer to a challenge put forward by Jan Łukasiewisz in 1926. He remarked that informal mathematical proofs do not take the form of Hilbert-style proofs. The main difference, according to Łukasiewisz, was that mathematicians tend to start from arbitrary suppositions, instead of axioms. He thus raised the problem of putting such a method in structural rules, and relating the so obtained system to the one of Hilbert.

It is probably partly due to his geographical position that we nowadays call Jaśkowski's system "Fitch deductions". As so many Polish articles of the time [61], it seems that Jaśkowski's paper has not succeeded in conquering a place in the view of the history of logic as held by today's logicians.

Luckily, Frederic Fitch (1908-1987), an American logician, has made extensive use of a system in form very similar to Jaśkowski's in his textbook "Symbolic Logic" [26]. This book did not just popularise Jaśkowski's ideas. In fact, it treats a wide range of ideas and the formal system Fitch introduces has no less intention than to "be adequate for all of mathematics essential to natural sciences". The form of the proof is not brought as the main topic; when he refers to Jaśkowski, Fitch says that he merely uses his system for its "pedagogical advantages". Because of the purpose of the book, Fitch's treatment is less formal and less precise than Jaśkowski's paper. Thanks to Fitch, Jaśkowski's system is still widely used in logic education.

The other paper introducing natural deduction, Gentzen's "Untersuchung

[1] *Studia Logica* 1 (1934), pp. 5–32.
[2] *Mathematische Zeitschrift* 39 (1935), 176–210, 405–431.

über das logische Schliessen" [27] awaited a different future. His motivation was very similar: setting up a formal system which comes as close as possible to actual reasoning. Just as in Jaśkowski's system, this includes suppositional reasoning. As we will see in the following sections this is one of the few properties that these systems have in common.

After introducing his natural deduction system, Gentzen continues his paper by proving the *Hauptsatz* for classical logic on the one hand, and for minimal logic and intuitionistic logic on the other hand: Every deduction in these systems can be reduced to a certain normal form. This normal form is free from "conceptual detours". In order to prove the Hauptsatz, Gentzen introduces yet another formal system, the so called *sequent calculus*. He then proves *cut-elimination* for sequent calculus, a property that entails the Hauptsatz for natural deduction.

In other words, the Hauptsatz presents a way to remove "conceptual detours". This does not mean that by eliminating these detours, the proof becomes shorter. Rather the contrary: The size of the proofs sometimes explodes as a result of normalisation. This holds both for normalisation of natural deduction and for cut-elimination in sequent calculus. Alessandra Carbone [15] [16] has developed so-called *flow-graphs* to study the blow-up caused by cut-elimination in sequent calculus in the last decade of the 20th century.

Besides the study into the complexity of proofs, the Hauptsatz brought forward other research too. The Hauptsatz for classical logic started a new branch in logic, which searches for an amendment of the rules for which the theorem can be proven *directly*, i.e. without the use of the sequent calculus. The Hauptsatz for intuitionistic logic has obtained a whole new meaning in the light of computability theory, especially in relation with λ-calculus.

Dag Prawitz [70] [71] was the first to prove the Hauptsatz directly. For minimal and intuitionistic logic he has succeeded in this without making important changes to Gentzen's system of natural deduction. In the case of classical logic, the situation was different. To prove the theorem Prawitz has removed $\vee$ and $\exists$ as primitive symbols. This has no consequences for the expressiveness and the strength of the system, because these connectives can be reintroduced as defined ones. However, it affects of course the structure of the proofs.

Since then other suggestions towards a direct proof of the Hauptsatz for classical logic have been proposed. These proposals have in common that they keep $\vee$ and $\exists$ as primitive symbols and many of them allow *multiple conclusions* (Kneale [50], Shoesmith and Smiley [78]). Finally Richard Statman was the first to prove the Hauptsatz for classical logic directly [80]. He did so by graph theoretical means, without allowing multiple conclusions.

This was not the end of multiple conclusion logic. The idea of making the rules of natural deduction more symmetric has some interest on its own account. Moreover, proof nets (by Girard [36]), a graphical presentation of the substructural linear logic, gained popularity. This formal system, which will be discussed in Chapter 5 and hence will be omitted in this overview, also permits multiple conclusions. Methods developed for proof nets have led to progress in the classical multiple conclusion logic, for example in the form of N-graphs [38].

The Hauptsatz for intuitionistic logic, on the other hand, obtained a new

meaning through its correspondence with λ-calculus. Church [17] and Kleene [48] have introduced the λ-calculus to describe formally the notion of computable function and to provide a foundation for logic and mathematics. Reduction of λ-calculus terms hence models the idea of computation. The λ-calculus has found an important application in functional programming languages.

The correspondence between the Hauptsatz and λ-calculus was first explicated by Howard [41], but it seems that Curry had remarked the resemblance between the construction of λ-calculus terms and the construction of propositional proofs already earlier (see [41]). William Tait had discovered the correspondence between cut-elimination and reduction of λ-terms [82].

The following sections discuss briefly the already mentioned formal systems: We give examples and point out some interesting properties. Most information can be found back in the survey articles [69] and [38], which we have gratefully used.

The Outlook (Section 1.6) introduces *Deduction Graphs*, the formalism that Part I of this thesis is about. By discussing our motivation and by pointing out some of their properties, we clarify their position in relation to the formal systems that already existed.

1.2 Formal Systems

1.2.1 Jaśkowski

Stanisław Jaśkowski answered in his paper "On the Rules of Suppositions in Formal Logic" [45] to a question raised by Jan Łukasiewisz in 1926. He had noticed that mathematicians often start proofs by postulating arbitrary suppositions and then start reasoning to see where they get. Hilbert-style systems do not describe this kind of suppositional reasoning: There every assumption has to be an axiom. Łukasiewisz thus raised the problem of putting such a suppositional method in structural rules and relating the so obtained system to the one of Hilbert.

Figure 1.1 shows a proof of $(\neg p \rightarrow \neg q) \rightarrow (q \rightarrow p)$ in one of Jaśkowski's systems. The proof consists of numbered lines preceded by the name of the system: here "td". This abbreviation stands for "theory of deduction" and is a classical proposition calculus without axioms. Other systems that are defined in the paper, are systems for classical proposition logic with axioms (TD), for minimal proposition logic without (ptd) or with (PTD) axioms, for intuitionistic logic without (itd) or with (ITD) axioms, for second order predicate logic with universal quantification (etd), and for first order predicate logic with universal quantification (cf).

Then follows a sequence of numbers, possibly the letter S (indicating that what follows is a supposition), and a formula in Polish notation: Np stands for $\neg p$, and Cpq stands for $p \rightarrow q$. The advantage of Polish notation is that brackets are not needed.

The sequence of numbers specifies the scope of the assumption, i.e. under

which assumptions a certain formula holds. On line 4, for example, we have 1.1.1.Nq, which indicates that Nq holds under the assumptions $CNpNq$ (line 1), q (line 2), and Np (line 3). In this example every subordinate proof contains at most one other subordinate proof. This is reflected by the number 1 in the sequences. If a subordinate proof would contain more than one other subordinate proof, the last number in the sequence of the second one would be 2, etc.

Finally, there is a motivation: That is a roman number, referring to one of the, in this case, four rules of the system, and an Arab number pointing to the lines it uses.

$td\,1$	$1.SCNpNq$	I
$td\,2$	$1.1.Sq$	I
$td\,3$	$1.1.1.SNp$	I
$td\,4$	$1.1.1.Nq$	III $1, 3$
$td\,5$	$1.1.p$	IV $3, 2, 4$
$td\,6$	$1.Cqp$	II $2, 5$
$td\,7$	$CCNpNqCqp$	II $1, 6$

Figure 1.1: A derivation in Jaśkowski's system.

In an earlier paper [44] Jaśkowski used boxes to border the scope of an assumption (see Figure 1.2). There, assumptions are not marked by an S and the motivation is omitted. Every line that is used in a box, should be in the box itself. Reiteration is used to get a line in the box in which it is needed.

1.2.2 Fitch

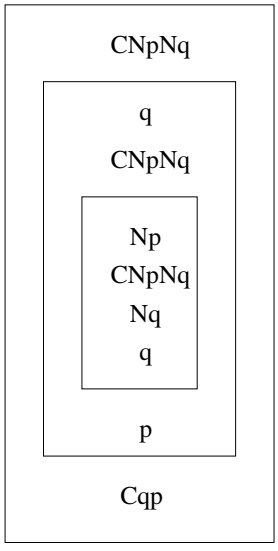
Frederic Fitch uses Jaśkowski's formalism in his textbook "Symbolic Logic" [26], be it in a somewhat streamlined form (Figure 1.3). The sequence of numbers has is replaced by vertical bars and the assumptions are highlighted by underlinings, the so-called "flags". The numbers of the rules are replaced by names, which makes the motivation easier to understand.

Fitch does not only change the way rules are referred to, he also changes the rules themselves. He adds even new rules until he has 42 of them, which he claims to "be adequate for all of mathematics essential to natural sciences". He also proves his system consistent.

The similarities with the systems of Jaśkowski lie thus foremost in the look of the proofs, and not in the actual logical strength.

1.2.3 Gentzen

In the same year as Jaśkowski, Gentzen also published a paper about natural deduction [27]. His aim was also to come closer to the way mathematicians reason than the already existing formal systems.



CCNpNqCqp

Figure 1.2: A derivation in Jaśkowski's earlier system with boxes.

1	$p \supset [q \supset r]$	hyp
2	$\quad p\&q$	hyp
3	$\quad p$	2, conj elim
4	$\quad q$	2, conj elim
5	$\quad p \supset [q \supset r]$	1, reit
6	$\quad q \supset r$	3,5, m p
7	$\quad r$	4,6, m p
8	$[p\&q] \supset r$	2-7, imp int

Figure 1.3: A derivation of Fitch.

Gentzen's system of natural deduction therefore includes suppositional reasoning, just like Jaśkowski's. In other respects the systems are quite different: Gentzen's deductions are trees and the scopes of the assumptions are not explicit. Furthermore, all rules are of a special form: they are either the introduction rule or the elimination rule of one of the connectives.

About the tree-structure, Gentzen says that it deviates from actual reasoning

in two ways: Firstly actual reasoning is necessarily linear, due to the linear ordering of utterances, and secondly in actual reasoning results may be applied repeatedly, whereas trees permit only one use of a derived formula. He sticks to the tree-structure because it makes the concept of a derivation more convenient and he finds the two deviations not essential.

The $\forall$-introduction rule of Gentzen is as follows:

$$\frac{\varphi}{\forall x\ \varphi[a := x]}$$

and has the side-condition that the *eigenvariable* a must not occur in any formula upon which φ depends.

Nowadays we often use the following rule for $\forall$-introduction instead of the original one:

$$\frac{\varphi}{\forall x\ \varphi}$$

with the condition that x does not occur in any of the non-discharged assumptions on which φ depends.

Gentzen starts by defining natural deduction for intuitionistic first-order predicate logic. From that he obtains classical first-order predicate logic by introducing the axioms $\varphi \vee \neg\varphi$.

Figure 1.4 shows a proof in Gentzen's system. The formulas $\forall y\ Fay$ and $\exists x \forall y\ Fxy$ are assumed at first. Then they are discharged at the steps $\exists$-E_1 and $\supset$-I_2 respectively.

$$\dfrac{\dfrac{\exists x\ \forall y\ Fxy \qquad \dfrac{\dfrac{\dfrac{1}{\forall y\ Fay}\,\forall\text{-E}}{Fab}}{\exists x\ Fxb}\,\exists\text{-}I}{\forall y\ \exists x\ Fxy}\,\forall\text{-I}}{\forall y\ \exists x\ Fxy}\,\exists\text{-}\mathrm{E}_1}{(\exists x\ \forall y\ Fxy) \supset (\forall y\ \exists x\ Fxy)}\,\supset\text{-}\mathrm{I}_2.$$

Figure 1.4: A proof in Gentzen's natural deduction formalism.

1.3 Cut-Elimination

1.3.1 Gentzen

Gentzen continues his paper treating the *Hauptsatz*: "No concepts enter into a proof other than those contained in its final result, and their use was therefore essential to the achievement of that result". At this stage Gentzen does not

make the Hauptsatz much more precise than this. In order to make this idea explicit, he introduces new systems, the so-called *sequent calculi*.

A sequent is a finite sequence of assumptions, followed by an arrow, or, in modern notation, a turnstyle ($\vdash$), and a conclusion (in the intuitionistic case) or a finite sequence of conclusions (in the classical case).

These calculi do not work with arbitrary assumptions, and instead of having an introduction-rule and an elimination-rule for each connective, they have left- and right-introduction rules for each of them. Furthermore there is an axiom rule and a cut-rule, the latter one running as follows (in modern notation):

$$\frac{\Gamma \vdash \Delta, A \qquad A, \Theta \vdash \Sigma}{\Gamma, \Theta \vdash \Delta, \Sigma}$$

Apart from these logical rules, there are also structural rules, which take care of the bookkeeping.

It is then proven that every sequent that can be derived from these rules, can also be derived without using the cut-rule. This is the theorem Gentzen uses as the formal definition of the Hauptsatz, and is also called *cut-elimination*.

Figure 1.5 shows a derivation of the law of excluded middle in the classical system.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \vdash A}{\vdash A, \neg A}\,\neg\text{-IS}}{\vdash A, A \vee \neg A}\,\vee\text{-IS}}{\vdash A \vee \neg A, A}\,\text{Interchange}}{\vdash A \vee \neg A, A \vee \neg A}\,\vee\text{-IS}}{\vdash A \vee \neg A}\,\text{Contraction}$$

Figure 1.5: A proof in Gentzen's sequent calculus.

From cut-elimination it follows that these systems are *decidable*.

1.3.2 Carbone

The proof of cut-elimination describes a way to get rid of the cuts by eliminating them one by one. During a cut-elimination one cut disappears, but it may also happen that several new ones are introduced. These applications of the cut-rule are applied to formulas of lower rank than the original.

Figure 1.6 shows schematically a derivation with a cut and the same derivation after eliminating that cut. The derivation D_1 gets copied, which causes an expansion of the derivation.

More than half a century after Gentzen, Alessandra Carbone uses *flow-graphs* [15, 16] (compare S. Buss [14]) to study these blow-ups of sequent calculus derivations. Flow-graphs are sequent calculus derivations of which the occurrences of atomic formulas are connected according to a certain algorithm. Figure

$$\dfrac{\dfrac{D_1}{\Delta \vdash A} \qquad \dfrac{\dfrac{D_2}{A,\Gamma \vdash B} \qquad \dfrac{D_3}{A,\Gamma \vdash C}}{A,\Gamma \vdash B \wedge C}}{\Delta,\Gamma \vdash B \wedge C}\ \text{Cut}$$

$$\dfrac{\dfrac{\dfrac{D_1}{\Delta \vdash A} \qquad \dfrac{D_2}{A,\Gamma \vdash B}}{\Delta,\Gamma \vdash B}\ \text{Cut} \qquad \dfrac{\dfrac{D_1}{\Delta \vdash A} \qquad \dfrac{D_3}{A,\Gamma \vdash C}}{\Delta,\Gamma \vdash C}\ \text{Cut}}{\Delta,\Gamma \vdash B \wedge C}$$

Figure 1.6: Expansion of a sequent calculus proof after cut-elimination.

1.7 shows such a flow-graph. We see that p and $\neg p$ are not connected and we can thus change for example $\neg p$ in r without losing the validity of the proof .[3]

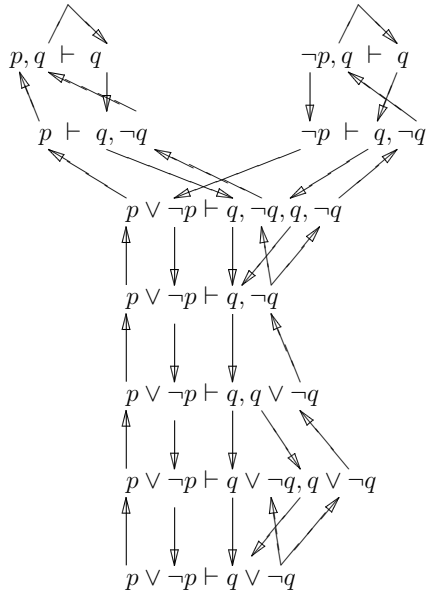


Figure 1.7: Flow-graph.

By means of graph-theoretical notions, like cycles, Carbone analyses in which cases exactly cut-elimination causes an enlargement of the derivation.

She uses a slight variation on the sequent calculus rules of Gentzen.

[3]This example has been taken from [38].

1.3.3 Prawitz

Prawitz gives in his paper "Ideas and results in proof theory" [71] and in his book "Natural deduction" [70] a direct proof of the Hauptsatz for natural deduction. He describes the natural deduction analogue of a cut, which is the introduction of a connective followed by the elimination of the same connective. To arrive at the same decidability result as Gentzen, Prawitz has to consider also so-called *commuting cuts*. Eliminating a commuting cut can be seen as removing the rules applied between an introduction and an elimination of the same connective, by changing the order of the applications. Then he shows how to eliminate cuts without going to the sequent calculus, so by transforming natural deduction derivations directly.

Figure 1.8 shows schematically the transformation for a $\rightarrow$-cut. $[A]^1$ means that the assumption A is discharged at the step marked with 1.

$$\dfrac{\dfrac{D_1}{A} \qquad \dfrac{\begin{array}{c}[A]^1 \ldots [A]^1 \\ D_2 \\ B\end{array}}{A \rightarrow B}\,1}{B} \qquad\qquad \begin{array}{c} D_1 \quad D_1 \\ A \ldots A \\ D_2 \\ B \end{array}$$

Figure 1.8: Elimination of a cut in natural deduction.

Prawitz has proven the Hauptsatz for intuitionistic logic without any adaptations of Gentzen's system for natural deduction [70][71]. For classical logic, however, he removes $\vee$ and $\exists$ as primitive symbols and reintroduces them as defined ones. This does not affect the expressiveness or logical strength of the system, but it does change the structure of the derivations.

1.3.4 Statman

Not everyone was satisfied with Prawitz's proof of the Hauptsatz for classical logic. A potential way to attack the problem and keep $\vee$ and $\exists$ as primitive connectives, is to make the rules of the calculus more symmetric by allowing multiple conclusions. (See Section 1.4).

Richard Statman approaches the matter differently in [80]. He remarks that Gentzen's natural deduction trees reflect only partly the structure of the derivation. They capture the relation R, where Rab means "a is a premiss of the inference with conclusion b", but cancellations in derivations do not add anything to the geometrical form.

He therefore draws derivations as graphs and puts also edges to capture the relation "a is an assumption occurrence cancelled by the inference with conclusion b". Figure 1.9 shows an example. The addition of this kind of edges results in the graph being not necessarily planar.

He then defines the complexity of proofs by topological means. One of his results is strong normalisation for full classical logic, i.e. with $\vee$ and $\exists$ as

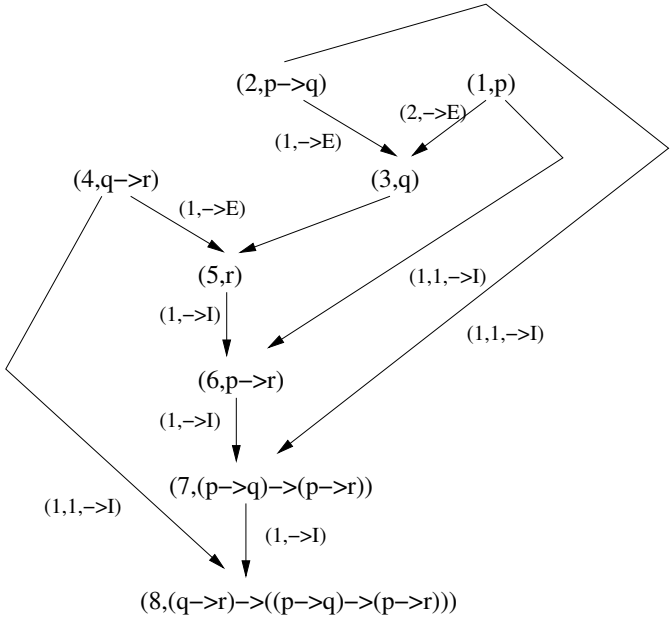


Figure 1.9: A derivation in Statman's system.

primitive symbols (see also [38]).

1.4 Multiple Conclusion Calculi

As mentioned before, multiple conclusion calculus started out as a potential way to prove the Hauptsatz directly without removing $\vee$ and $\exists$ as primitive symbols. The idea is to use for example the following rule for $\vee$-elimination:

$$\frac{P \vee Q}{P \quad Q}$$

instead of

$$\frac{\begin{array}{ccc} & P & Q \\ & \vdots & \vdots \\ P \vee Q & C & C \end{array}}{C}$$

The new rule is symmetric to the $\wedge$-introduction, which runs:

$$\frac{P \quad Q}{P \wedge Q}$$

It is important to notice that, although proofs done in a multiple conclusion logic do in general not have a tree structure, they do not meet the properties

of actual reasoning from which Gentzen deviated either: The proofs are neither linear, nor do they permit to use results repeatedly.

Figure 1.10 shows a *table of development* as defined by Kneale [50]. It shows that $P \to Q$ follows from $\neg P$. Informally this table can be understood as follows. Suppose $\neg P$. From this we may conclude P or $P \to Q$ (because from every formula we may conclude P or $P \to Q$). However, P gives a contradiction with the assumption $\neg P$, from which we may also conclude $P \to Q$. Hence $P \to Q$ follows from $\neg P$.

To get a valid table of development, it is not enough to keep to the local logical rules. A table of development should also satisfy a global condition: Formulas that are already connected through one or more horizontal lines, must not be connected again. In this way cycles are avoided.

$$\dfrac{\neg P \qquad \dfrac{P \qquad P \to Q}{P \to Q}}{P \to Q}$$

Figure 1.10: Table of development.

Shoesmith and Smiley [78] change the tables of development into *graph arguments* (Figure 1.11; the intuitive explanation of this as a logical derivation is the same as for Figure 1.10). They use special symbols to mark premisses (a triangle with the base on top), conclusions (a triangle with the base at the bottom) and other formula occurrences (a circle). In the example graph it is allowed to change the triangle associated to one of the premisses $\neg P$ into a circle, to express that the proof really depends on just one premiss.

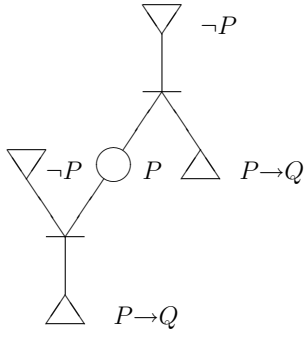


Figure 1.11: Graph argument.

A similar result can be obtained by identifying premisses. Note, however, that sharing introduces cycles. This means that we need an altered version of the global condition.

Graphs that look more like natural deduction derivations, *N-graphs*, are introduced by Grisi de Oliveira and de Queiroz [38]. This formalism allows again suppositional reasoning, a property that was lost in the other multiple conclusion systems. Discharging of a hypothesis is indicated by a so-called meta-edge (see Figure 1.12).

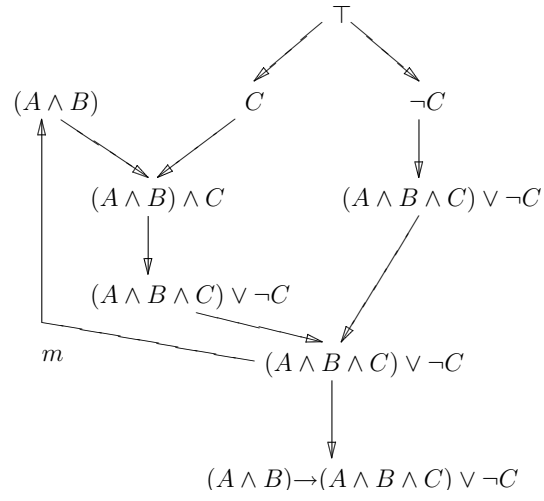


Figure 1.12: N-graph.

This system provides rules for contraction and expansion, but still it cannot be said that it fulfills the property of actual reasoning that results that are already obtained may be used repeatedly. This is mainly due to the fact that conclusions of the graph should be understood as *disjunctive*, and not as conjunctive. A conclusion of a graph cannot be seen as a result on itself.

Figure 1.13, for example, can be interpreted as a proof of $(A \wedge B) \vee (\neg A \vee \neg B)$. So, $A \wedge B$ is not claimed to be true.

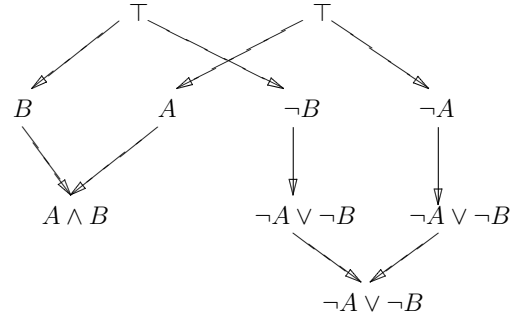


Figure 1.13: N-graph with multiple conclusions.

The global soundness criterion of N-graphs is inspired by a global soundness

criterion of proof nets.

1.5 λ-calculus

The Hauptsatz for intuitionistic logic obtained a new meaning through its correspondence with the λ-calculus.

Church [17] and Kleene [48] have introduced the λ-calculus to describe formally the notion of computable function and to provide a foundation for logic and mathematics. In this formalism the term $\lambda x.\lambda y.x$, for instance, represents a function that takes two arguments and yields the former.

We give here the definition of the *simply typed λ-calculus*. Typed λ-calculi are a refinement of the more general untyped λ-calculi. A term in a typed λ-calculus can be interpreted as a function with a specified domain and codomain.

Definition 1.5.1 *We start out with a countable number of type-variables and for every type a countable number of term-variables. We will use α to range over the type-variables and for all types σ we will use x^σ to range over the term-variables. The types and terms are defined as follows:*

$$\begin{array}{lll} \textit{Types } \sigma,\tau & ::= & \alpha \mid \sigma{\to}\sigma \\ \textit{Terms } M,N & ::= & x^\sigma \mid (MN) \mid \lambda x^\sigma.M \end{array}$$

The typing rules for terms are defined as follows. Let Γ be an environment containing term-variable declarations. The judgment $\Gamma \vdash M{:}\sigma$ states that M is a term of type σ.

$$\Gamma \vdash x^\sigma{:}\sigma \quad \textit{if } x^\sigma{:}\sigma \in \Gamma \qquad\qquad \frac{\Gamma \vdash M{:}\sigma{\to}\tau \quad \Gamma \vdash N{:}\sigma}{\Gamma \vdash (MN){:}\tau}$$

$$\frac{\Gamma, x^\sigma{:}\sigma \vdash M{:}\tau}{\Gamma \vdash \lambda x^\sigma.M{:}\sigma{\to}\tau}$$

From the way we have presented the typing rules, the correspondence between types and formulas of minimal logic, and between terms and natural deduction derivations, is clearly suggested. We have not used exactly Gentzen's notation for natural deduction here: We take along the assumptions through the proof. This simplifies the correspondence a bit, but it is not essential.

The conformity does not stop here. As λ-calculus represents the theory of computable functions, it provides a notion of calculation, the so-called *β-reduction* on terms:

$$(\lambda x^\sigma.M)N \quad \longrightarrow_\beta \quad M[x^\sigma := N]$$

This turns out to agree nicely with cut-elimination.

The correspondence between types and formulas, terms and proofs, and β-reduction and cut-elimination, is called the *Curry-Howard-de Bruijn isomorphism*.

1	$A{\to}A{\to}B$		1	$(A{\to}B){\to}A$	
2	$(A{\to}B){\to}A$		2	$A{\to}A{\to}B$	
3	A		3	A	
4	$A{\to}B$	$\Rightarrow$E, 1, 3	4	$A{\to}B$	$\Rightarrow$E, 2, 3
5	B	$\Rightarrow$E, 4, 3	5	B	$\Rightarrow$E, 4, 3
6	$A{\to}B$	$\Rightarrow$I, 3, 5	6	$A{\to}B$	$\Rightarrow$I, 3, 5
7	A	$\Rightarrow$E, 2, 6	7	A	$\Rightarrow$E, 1, 6
8	B	$\Rightarrow$E, 6, 7	8	B	$\Rightarrow$E, 6, 7

Figure 1.14: Distinct Fitch-style natural deduction derivations.

Howard was the first to describe this in his paper "The Formulae-as-Types Notion of Construction" [41]. In this paper he explicates the isomorphism for full intuitionistic predicate logic. He claims that Curry had already noticed the correspondence for minimal logic. The resemblance between β-reduction and cut-elimination stems from a paper of William Tait [82] on Gödel's alternative for Gentzen's proof of the consistency of number theory.

The λ-calculus has found an important application in programming languages.

1.6 Outlook

Now that we have seen some formalisms and we have placed them in a context, we discuss briefly the formalism of *deduction graphs*, the subject of Part I of this thesis.

Fitch-style natural deduction allows the re-use of subproofs. Moreover, it represents reasoning in a linear way. The linearity of the proofs may indeed reflect actual reasoning, at the same time it seems that the order in which the propositions are written down, is not essential to the argumentation.

Figure 1.14 shows for example two Fitch-style derivations which only differ in the order of the flags. (We use here the notation of [85]; see also the appendix). Is this enough to consider them as distinct?

In [31] a Curry-Howard-de Bruijn correspondence is obtained by ignoring this kind of non-essential differences between Fitch-style derivations.

Gentzen-style natural deduction introduces no such inessential differences. At the same time, subproofs cannot be shared, which amounts to making the same subderivation several times (see Figure 1.15).

Now the question arises whether we could allow the repeated use of subresults

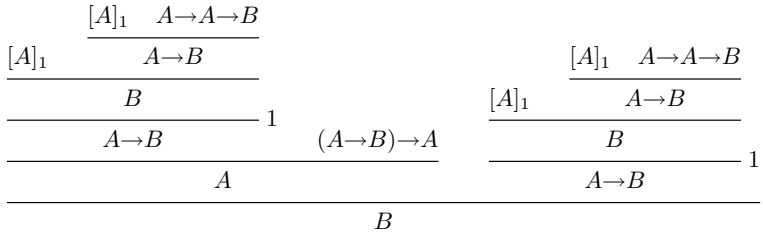


Figure 1.15: Gentzen natural deduction with the same subderivation twice.

without introducing non-essential distinctions. What we would like is something like in Figure 1.16, where the proof of $A \to B$ gets shared.

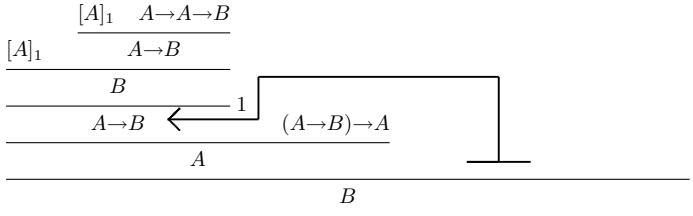


Figure 1.16: Natural deduction with shared subderivation.

This sharing is not totally unproblematic. A derivation like in Figure 1.17 is not acceptable, because this proof of $A \to B$ is obtained under the assumption A, and the "second use" of the subresult is out of scope of this assumption.

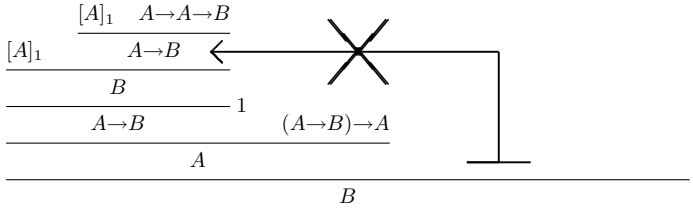


Figure 1.17: Natural deduction with incorrectly shared subderivation.

We thus propose the formalism of *deduction graphs* to deal correctly with the sharing of subproofs by making the scope of an assumption explicit. Figure 1.18 shows an example of a deduction graph.

The use of subresults is depicted by edges. The edges point from the conclusion to the premisses, because, just like in Fitch-style deduction, we refer *back*. Furthermore, in case a derivations contains no sharing and has just one conclusion, we obtain an arborescence, which we would not have obtained if we let the arrows point from the premisses to the conclusion.

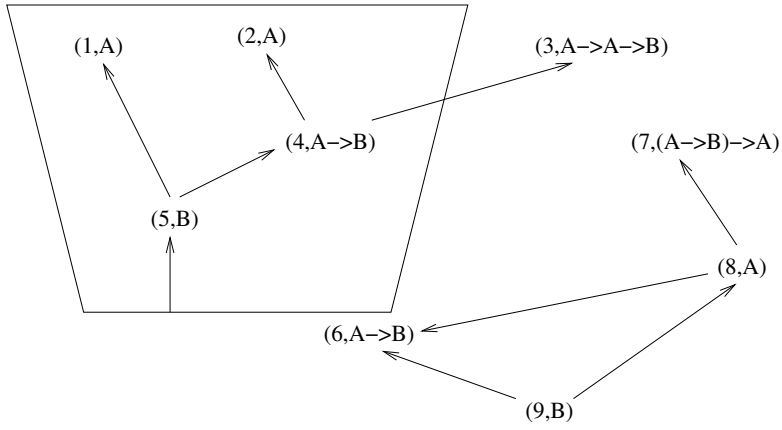


Figure 1.18: Deduction graph.

The direction of the edges is not the only difference with Statman's graphs. His graphs have edges to reflect the cancellation relation, whereas deduction graphs use *boxes* to make scopes explicit. The boxes are not in the first place to describe the geometrical structure of natural deduction, but to control *sharing*, a feature absent in both Gentzen's natural deduction and Statman's graphs.

The scope of an assumption is drawn as a box. Severe restrictions on edges that may enter a box can now take care of correct sharing. We forbid for instance an edge from $(8, A)$ to $(4, A{\rightarrow}B)$, instead of the edge to $(6, A{\rightarrow}B)$: Such edges may not enter a box. This use of boxes is reminiscent of Jaśkowski's original idea. His derivations, however, are linear. Furthermore, in proof nets of multiplicative exponential linear logic (MELL) global logical steps are also depicted by boxes. In Chapter 5 we go deeper into the relation between deduction graphs and proof nets.

The deduction graph formalism is a multiple conclusion logic: There may be several nodes without incoming edges. Unlike the other multiple conclusion logics, the conclusions of the graph should be understood in a *conjunctive* way.

Allowing sharing in derivations hinders cut-elimination. In Figure 1.18 it is not immediately clear how to eliminate the cut formed by the $\rightarrow$-introduction followed by the $\rightarrow$-elimination: The node $(6, A{\rightarrow}B)$ may not be removed, as not only $(9, B)$ depends on it, but also $(8, A)$.

The study of deduction graphs is thus for an important part centered around cut-elimination, including Curry-Howard-de Bruijn morphisms.

Chapter 2

Basic Notions

2.1 Orderings and graphs

2.1.1 Partial orderings and linear orderings

In the next section (Section 2.1.2) a structure is defined with two partial orderings on its nodes. Those partial orderings should, when combined, yield again a partial ordering. This is important, because linear extensions of the latter partial ordering give us a way to pass through the nodes of the structure in a significant way.

This section treats some basic results in the theory of orderings. It gives some elementary definitions, and we will explain what is meant by "combining two partial orderings". Identifying linear orderings with permutations, we then land at Theorem 2.1.12, which is a refinement of the well-known theorem of permutation groups that says that any permutation can be written as a product of neighbouring permutations. This refinement relates the various linear extensions of a partial ordering to each other in a step-by-step way: When two neighbouring elements get interchanged before applying the permutation, the result may or may not be a linear extension of the partial ordering again. The Theorem tells us, that we can obtain any linear extension of a partial ordering from any other one by interchanging several times neighbouring elements before applying the latter permutation and keeping a linear extension of the given partial ordering at each stage.

The orderings we consider are decidable orderings on finite sets, unless specifically mentioned otherwise.

Definition 2.1.1 (Transitive Closure) *Let R be a binary relation. We will write $\mathsf{CL}(R)$ for the* transitive closure of R. *So $\mathsf{CL}(R)$ is the smallest set such that:*

1. $R \subseteq \mathsf{CL}(R)$;
2. *If* $(a, b) \in \mathsf{CL}(R)$ *and* $(b, c) \in \mathsf{CL}(R)$, *then* $(a, c) \in \mathsf{CL}(R)$.

Definition 2.1.2 (Partial Ordering, Linear Ordering)
Let R be a binary relation on a set S. R is a partial ordering*, when:*

- $(a, a) \notin R$*, for all $a \in S$ (irreflexivity);*
- *If $(a, b) \in R$ and $(b, c) \in R$, then $(a, c) \in R$ (transitivity);*
- *If $(a, b) \in R$, then $(b, a) \notin R$. (anti-symmetry).*

R is a linear ordering*, when it answers the following additional requirement:*

- $(a, b) \in R$ *or* $(b, a) \in R$ *or* $a = b$ *(totality).*

We will use Greek letters for partial orderings and linear orderings.

Definition 2.1.3 (Restricted linear ordering) *If φ is a linear ordering of a set S and $T \subseteq S$, then $\varphi_{|T}$ is the relation φ* restricted *to T.*

Lemma 2.1.4 *Let φ be a linear ordering of a set S and $T \subseteq S$. Then $\varphi_{|T}$ is a linear ordering of the set T.*

Definition 2.1.5 (Linear Extension) *Let φ be a partial linear ordering of a set S. The ordering ψ is a* linear extension *of φ, when $\varphi \subseteq \psi$, and ψ is a linear ordering of S.*

Remark 2.1.6 *Note that every partial ordering has a linear extension.*

The following definition tells how to combine two relations. The idea is simply to join them and take the transitive closure of the result.

Definition 2.1.7 (Consistent Partial Orderings) *Let S be a set and let R_1 and R_2 be two relations on S. We say that R_1 is* consistent *with R_2, if $\mathsf{CL}(R_1 \cup R_2)$ is a partial ordering of S.*

Note that consistency is a symmetric property: if R_1 is consistent with R_2, then R_2 is consistent with R_1 too.

We give two simple properties of orderings (without proof):

Lemma 2.1.8

1. *If R is a subset of the linear ordering φ, then so is $\mathsf{CL}(R)$.*
2. *If φ is a linear ordering of the set S, then every transitive subset R of φ is a partial ordering of S.*

We have defined linear orderings on a set S as a set of tuples (Lemma 2.1.2) with some properties. Another way to look at it, is to consider a linear ordering as an enumeration of the elements of a set, that is: A bijective function from a subset of $\mathbb{N}$ of the right size to S.

Definition 2.1.9 (Permutation) *Let S be a set with n elements. A* permutation *on S is a bijection $\underline{n-1} \to S$, where $\underline{n-1} = \{0, 1, \ldots, n-1\}$.*

These two views on linear orderings lead essentially to the same notion:

Lemma 2.1.10 *There is a bijection between the linear orderings on S and the permutations of S.*

From now on φ will indicate the linear ordering as well as the associated permutation.

Next we define π_i^n, the permutation that interchanges the elements i and $i+1$.

Definition 2.1.11 *Let $n \in \mathbb{N}$ and let $i < n-2$. The permutation π_i^n is defined as follows:*

$$\pi_i^n(k) \begin{cases} k & \text{if } k \neq i, i+1 \\ i+1 & \text{if } k = i \\ i & \text{if } k = i+1 \end{cases}$$

We simply write π_i, when n is clear from the context.

Remember the classical theorem of symmetric groups that says that any permutation can be written as product of interchanges of neighbouring elements. From this we conclude that any permutation can be obtained from any other permutation by interchanging neighbouring elements on before hand: Suppose we want to obtain permutation ψ, starting from φ, then the theorem provides us the following equations for φ^{-1} and ψ:

$$\varphi^{-1} = \pi_{k_0} \circ \pi_{k_1} \circ \ldots \circ \pi_{k_n}$$

$$\psi = \pi_{l_0} \circ \pi_{l_1} \circ \ldots \circ \pi_{l_m}$$

From this we find:

$$\varphi \circ \pi_{k_0} \circ \pi_{k_1} \circ \ldots \circ \pi_{k_n} \circ \pi_{l_0} \circ \pi_{l_1} \circ \ldots \circ \pi_{l_m} = \psi$$

We are now interested in the sequence (φ), $(\varphi \circ \pi_{k_0})$, $(\varphi \circ \pi_{k_0} \circ \pi_{k_1})$, …, $(\varphi \circ \pi_{k_0} \circ \pi_{k_1} \circ \ldots \circ \pi_{k_n} \circ \pi_{l_0} \circ \pi_{l_1} \circ \ldots \circ \pi_{l_m})$. Suppose that both φ and ψ are linear extensions of the same partial ordering. Could we have chosen the π_{k_i}s and π_{l_j}s in such a way that every element of this sequence is again a linear extension of the partial ordering? The following theorem states that we can.

Theorem 2.1.12 *Let G be a set. Let α be a partial ordering of G. Suppose φ and ψ are linear extensions of α. Then there exist a k and $i_0, \ldots, i_k$ such that*

$$\varphi \circ \pi_{i_0} \circ \ldots \circ \pi_{i_k} = \psi$$

and:

$$\varphi \circ \pi_{i_0} \circ \ldots \circ \pi_{i_j} \text{ is a linear extension of } \alpha \text{ for every } j \leq k.$$

Proof Suppose $\varphi(m) = \psi(m)$ for all $m < n$ for some $n \in \mathbb{N}$. Take $k := \varphi^{-1}(\psi(n))$. Define the orderings $\zeta_0, \zeta_1, \ldots, \zeta_{k-(n+1)}$ by:

$$\zeta_0 = \varphi \circ \pi_{k-1}$$

$$\zeta_{p+1} = \zeta_p \circ \pi_{k-(p+2)}.$$

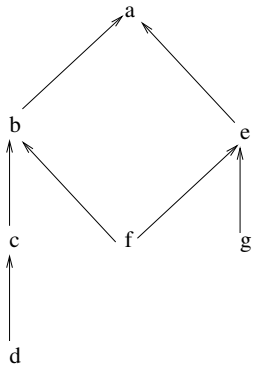
We see by induction that:

1. For all $i \leq k-(n+1)$, ζ_i is a linear extension of α.
2. For all $i \leq k-(n+1)$, for all $m < n$, $\zeta_i(m) = \varphi(m)$.
3. For all $i \leq k-(n+1)$, $\zeta_i(k-(i+1)) = \psi(n)$.

Hence $\zeta_{k-(n+1)}(m) = \psi(m)$ for all $m < n+1$. Thus the statement follows with induction.

Example 2.1.13 *Consider the following figure. It represents the partial ordering given by the transitive closure of the relation:*

$$\{(a,b),(a,e),(b,c),(b,f),(c,d),(e,f),(e,g)\}.$$



Let $\varphi := \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ a & b & c & d & e & f & g \end{pmatrix}$ *and* $\psi := \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ a & e & g & b & f & c & d \end{pmatrix}$. *We will rewrite* φ *in* ψ *by just interchanging adjacent nodes in the linear ordering and keeping a linear extension of the partial ordering at each stage.*

- *We start by placing* $\psi(1) = e$ *at the right position. We see that:*

$$\varphi \circ \pi_3 \circ \pi_2 \circ \pi_1 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ a & e & b & c & d & f & g \end{pmatrix}$$

- *We continue by placing* $\psi(2) = g$ *at the right position:*

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ a & e & b & c & d & f & g \end{pmatrix} \circ \pi_5 \circ \pi_4 \circ \pi_3 \circ \pi_2 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ a & e & g & b & c & d & f \end{pmatrix}$$

- *Now, we place* $\psi(4) = f$ *at the right position:*

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ a & e & g & b & c & d & f \end{pmatrix} \circ \pi_5 \circ \pi_4 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ a & e & g & b & f & c & d \end{pmatrix} = \psi$$

Thus, we have found that:

$$\varphi \circ \pi_3 \circ \pi_2 \circ \pi_1 \circ \pi_5 \circ \pi_4 \circ \pi_3 \circ \pi_2 \circ \pi_5 \circ \pi_4 = \psi$$

2.1.2 Closed box directed graphs

A closed box directed graph is a directed acyclic graph with labelled nodes, which may be contained in boxes. This structure we will use to represent a deduction, instead of, for example, trees in Gentzen-Prawitz style natural deduction. The idea is that the arrows express one-step derivability, whereas the boxes border the scope of a local hypothesis. As it is not allowed to use any hypothesis outside its scope, closed box directed graphs do not have arrows with a source outside a box, pointing inwards (unless it is from the node closing the scope). It is because arrows do not point into boxes, that we call this structure *closed* box directed graphs.

Figure 2.1 shows a closed box directed graph that represents a deduction (the left most picture).

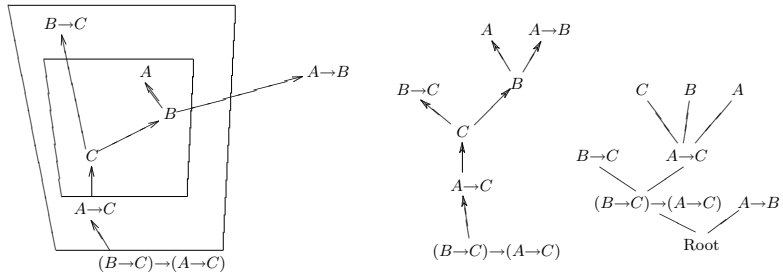


Figure 2.1: Deduction graph of $(B{\to}C){\to}(A{\to}C)$ from $A{\to}B$ and link and place graphs

This is slightly reminiscent of the bigraphs of Milner [62]. There also collections of nodes can form nodes themselves, giving rise to the view of a bigraph as a combination of a *link graph* and a *place graph*. The first describes the nodes and edges between them and the second describes the nesting structure. This terminology can be applied to our deduction graphs as well. The left most graph in Figure 2.1 then gives rise to the two graphs to the right of it. The connection with partial orderings as described in the previous section can now easily be observed.

Notwithstanding this superficial correspondence, our deduction graphs have a quite different nature from Milner's bigraphs: in bigraphs edges have no direction and can be multi-linked. Furthermore, nodes in bigraphs have a fixed *arity* which determines the number of edges it connects to. In our (initial) setting the number of ingoing edges is unlimited.

Definition 2.1.14 (Closed box directed graph) *A* closed box directed graph *is a triple* $\langle X, G, (\mathcal{B}_i)_{i\in I}\rangle$ *where* X *is a set of* labels, G *is a directed graph where all nodes have a label in* X *and* $(\mathcal{B}_i)_{i\in I}$ *is a collection of sets of nodes of* G*, the* boxes. *Each box* $\mathcal{B}_i$ *corresponds to a node, the* box node *of* $\mathcal{B}_i$. *Moreover, the boxes* $(\mathcal{B}_i)_{i\in I}$ *should satisfy the following properties:*

1. *(Non-overlap) Two boxes are disjoint or one is contained in the other:* $\forall i, j \in I(\mathcal{B}_i \cap \mathcal{B}_j = \emptyset \vee \mathcal{B}_i \subset \mathcal{B}_j \vee \mathcal{B}_j \subset \mathcal{B}_i)$,
2. *(Box-node edge) There is only one outgoing edge from a box-node and that points into the box itself (i.e. to a node in the box),*
3. *(No edges into a box) Apart from the edge from the box-node, there are no edges pointing into a box.*

Formally, the labeled directed graph G is a triple $\langle N, E, L\rangle$, where N is a set of nodes, E is a set of edges and L is a (labelling) function from N to X. Furthermore, there is an injection from the set of boxes to the set of nodes, allowing us to identify a box with its *box node*, so we freely write $\in$ for the transitive relation on nodes: $n \in m$ denotes that m is the box node of some box $\mathcal{B}$ and $n \in \mathcal{B}$. For example in the right most graph of Figure 2.2, $1 \in 6$, $4 \in 6$, but $5 \notin 6$, $6 \notin 6$. Following Milner's idea, we define the link graph and the place graph of a closed box directed graph.

Definition 2.1.15 (Link graph) *The* link graph *of a closed box directed graph* $\langle X, G, B\rangle$ *is the labeled directed graph* $\langle X, G\rangle$.

Definition 2.1.16 (Place graph)
The place graph *of a closed box directed graph* $\langle X, \langle N, E, L\rangle, B\rangle$ *is the labeled graph* $\langle X, \langle N, \in, L\rangle\rangle$, *where* $\in$ *is the membership relation between nodes and boxes.*

Definition 2.1.17 *A closed box directed graph* $\langle X', \langle N', E', L'\rangle, B'\rangle$ *is a* subgraph *of a closed box directed graph* $\langle X, \langle N, E, L\rangle, B\rangle$, *when:*

1. $X' \subseteq X$;
2. $\langle N', E'\rangle$ *is a subgraph of* $\langle N, E\rangle$;
3. $L' = L_{|N'}$;
4. $\langle N', \in\rangle$ *is the subgraph of* $\langle N, \in\rangle$ *induced by* N'.

We denote a closed box directed graph by $\langle G, (\mathcal{B}_i)_{i\in I}\rangle$, or by G, leaving the set of labels X, and possibly also the set of boxes, implicit.

So, if $\mathcal{B}$ is a box, $m \in \mathcal{B}$, $n \notin \mathcal{B}$ and $n \longrightarrow\!\triangleright\, m$ then n is the box-node of $\mathcal{B}$. Moreover, if $\mathcal{B}$ is a box with box-node n, then there is exactly one edge from n to some $m \in \mathcal{B}$.

In Figure 2.2 we give three non-examples of closed box directed graphs and one example of a closed box directed graph. (For clarity, we have left out the labels of the nodes).

Definition 2.1.18 (Box-topological ordering) *Let* G *be a closed box directed graph. A* box-topological ordering *of* G *is a linear ordering of the nodes of* G, *that both contains the link graph of* G *and its place graph.*

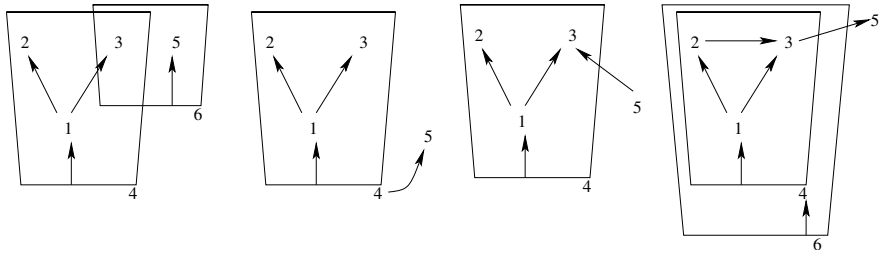


Figure 2.2: Three non-examples and one example of a closed box directed graph

Example 2.1.19 *Consider the closed box directed graphs of Figure 2.3. The link-graph imposes the following partial ordering:* $1 < 2 < 3 < 4$. *The box-graph, on the other hand, imposes the partial ordering given by:* $1 < 2$ *and* $4 < 2$. *We conclude that no box-topological ordering of this graph exists, as both* $(2, 4) \in R$ *and* $(4, 2) \in R$, *according to any relation* R *that includes the place graph and the link graph.*

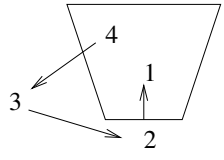


Figure 2.3: A closed box directed graph without box-topological ordering.

The following lemma relates the box-topological ordering of a graph G to its link graph and its place graph.

Lemma 2.1.20 *Let* G *be a closed box directed graph. Let* L *be the link graph and let* P *be the place graph. Then:*

$$G \text{ has a box-topological ordering} \Leftrightarrow L \text{ is consistent with } P.$$

Proof

$\Rightarrow$ Suppose ζ is a box-topological ordering of G. Then $L \cup P$ is a subset of ζ. Then also $\mathsf{CL}(L \cup P)$ is a subset of ζ. Thus ζ is a linear extension of $\mathsf{CL}(L \cup P)$.

$\Leftarrow$ The relation L is consistent with P, so $\mathsf{CL}(L \cup P)$ is a partial ordering. Then $\mathsf{CL}(L \cup P)$ has a linear extension, say ζ. So ζ is a linear ordering of the nodes of G, and it contains L and P.

In a directed graph with boxes, we want to define which nodes are "visible" and can henceforth be used to derive new (logical) conclusions. We therefore define the notion of *scope*, which is not related to the graph structure, but to the boxes.

Definition 2.1.21 *Let $\langle G, (\mathcal{B}_i)_{i\in I}\rangle$, be a closed-box directed graph and let n_0 and n_1 be nodes in this graph.*

- *Node n_1 is* in scope of *n_0 if n_0 is in all boxes that n_1 is in. In a formula: $\forall i \in I(n_1 \in \mathcal{B}_i \Rightarrow n_0 \in \mathcal{B}_i)$. (So the nodes in scope of n_0 are the nodes that are in 'wider' boxes.)*
- *The nodes n_0 and n_1 are* at the same depth*, when n_0 is in scope of n_1, and n_1 is in scope of n_0. Node n_0 is* at a greater depth *than n_1, when n_1 is in scope of n_0, but n_0 is not in scope of n_1.*
- *Node n_1 is a* top-level node *if n_1 is not contained in any box.*
- *The* free nodes *are the top-level nodes that have no outgoing edges.*
- *The* free nodes of a box $\mathcal{B}$ *are the nodes in $\mathcal{B}$ that become free nodes if we remove $\mathcal{B}$ and all boxes that contain $\mathcal{B}$.*

Definition 2.1.22 *Let G be a closed box directed graph with box-topological ordering φ. We will write $\widehat{\varphi}$. for the restriction of φ to top-level nodes.*

2.2 Deduction graphs

2.2.1 Definition of Deduction Graphs

We now define the deduction graphs that correspond to minimal proposition logic. We give an inductive definition, in the style of Gentzen and Prawitz, with the difference that now we are not dealing with a tree structure but with a graph structure, which makes the definition more involved.

First, we define the set of formulas of implicational logic.

Definition 2.2.1 (Formulas of implicational logic)
The set of formulas of implicational logic, Form *is defined as follows:*

1. *For every $i \in \mathbb{N}$, A_i is a formula of* Form*;*
2. *If A and B are two formulas of* Form*, then $(A{\rightarrow}B)$ is a formula of* Form *too.*
3. *The constructions 1 and 2 are the only ways to obtain a formula of implicational logic.*

Furthermore, we adopt the following convention for the brackets in formulas;

1. $\rightarrow$ is right-associative;

2. Outer brackets are not written, nor are any other brackets that do not contribute to the understanding of the formula.

So, for example, $A{\to}B{\to}C \equiv (A{\to}(B{\to}C))$.

Note that we will use arrows for three different purposes. For clarity we have used different arrows for each purpose:

- In the text we denote an edge by $\longrightarrow\!\!\triangleright$,
- We denote the implication by $\to$,
- We denote a reduction by $\longrightarrow\!\!>$.

Definition 2.2.2 *The collection of* deduction graphs for minimal proposition logic *is the set of closed box directed graphs over* $\mathbb{N} \times \mathsf{Form}$ *inductively defined as follows.*

Axiom *A single node* (n, A) *is a deduction graph,*

Join *If* G *and* G' *are disjoint deduction graphs, then* $G'' := G \cup G'$ *is a deduction graph.*

$\to$-E *If* G *is a deduction graph containing two nodes* $(n, A{\to}B)$ *and* (m, A) *at the top level, then the graph* $G' := G$ *with*

- *a new node* (p, B) *at the top level*
- *an edge* $(p, B) \longrightarrow\!\!\triangleright (n, A{\to}B)$,
- *an edge* $(p, B) \longrightarrow\!\!\triangleright (m, A)$,

is a deduction graph. (See Figure 2.4, the left part.)

$\to$-I *If* G *is a deduction graph containing a node* (j, B) *with no ingoing edges and a finite set of free nodes with label* A, $(n_1, A), \ldots, (n_k, A)$, *all at the top level, then the graph* $G' := G$ *with*

- *A box* $\mathcal{B}$ *with box-node* $(n, A{\to}B)$, *containing the nodes* (j, B) *and* $(n_1, A), \ldots, (n_k, A)$ *and no other nodes that were free in* G,
- *An edge from the box node* $(n, A{\to}B)$ *to* (j, B)

is a deduction graph under the proviso that it is a well-formed closed box directed graph. *(See Figure 2.4, the right part. The nodes* (n_1, A), $\ldots, (n_k, A)$ *are* discharged*)*

Repeat *If* G *is a deduction graph containing a node* (n, A) *at the top level, the graph* $G' := G$ *with*

- *a new node* (m, A) *at the top level,*
- *an edge* $(m, A) \longrightarrow\!\!\triangleright (n, A)$

is a deduction graph.

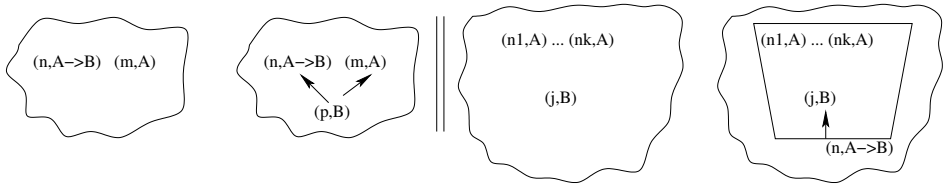


Figure 2.4: The →-E and →-I rule of deduction graphs

From now on, we will simply write "deduction graph" instead of "deduction graph for minimal proposition logic".

Example 2.2.3 *We give three simple deduction graphs in Figure 2.5. In (II), we see an example where no assumption is discharged in the introduction of the →. In (III), "garbage" (nodes* 2 *and* 3*) occurs inside a box – it is just an involved way of proving* $B \to B$*.*

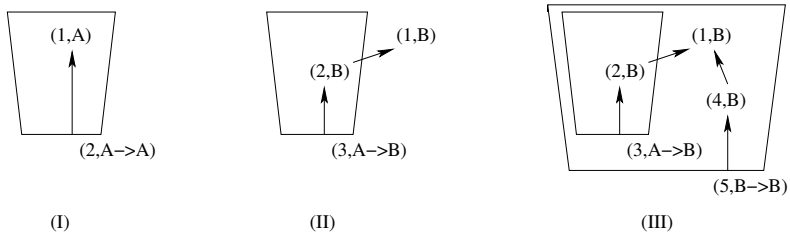


Figure 2.5: Three simple deduction graphs

Example 2.2.4 *We give another example of a deduction graph. In the Figure 2.6, the part* H *is some unspecified part of the graph that contains a node* $(7, A)$*. This deduction graph can be seen as a derivation of* B *from an assumption* $A \to A \to B$ *(assuming a derivation* H *of* A*).*

Lemma 2.2.5 *Every deduction graph is acyclic.*

Proof Number the nodes in the ordering in which they have been introduced. Note that Definition 2.2.2 only allows edges $m \longrightarrow\!\!\triangleright\, n$ if $m > n$.

Example 2.2.6 *If we ignore the non-depicted nodes of* H *in the deduction graph of Example 2.2.4 (Figure 2.6), the usual linear ordering of the naturals itself can be used to give a box-topological ordering. More generally, each linear ordering that satisfies the constraints* $7 < 8$, $4 < 5 < 6 < 8$, $1 < 5$, $2 < 4$, $3 < 4$ *is a box-topological ordering of the deduction graph.*

In the rightmost graph of Figure 2.2, the only possible box-topological ordering of the graph is $5 < 3 < 2 < 1 < 4 < 6$.

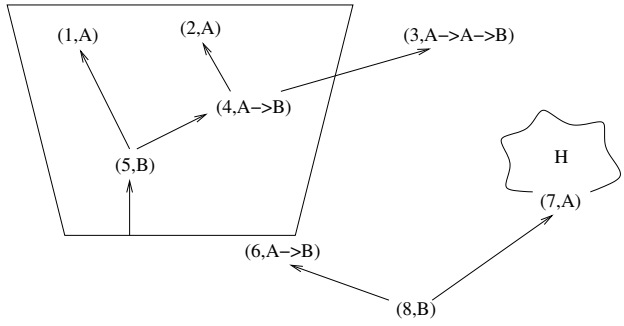


Figure 2.6: Example of a deduction graph

The following lemma enables us to prove that a given graph G is a deduction graph without explicitly supplying a construction. The lemma basically says that we just have to check that G has a box-topological ordering and that each node of G is of one of four possible types (A, E, I or R, to be defined in the lemma). These types correspond to the Axiom, Join, $\rightarrow$-I and $\rightarrow$-E construction cases of Definition 2.2.2. The lemma also makes it possible to define a function inductively on nodes of a deduction graph, independently of how the graph has been created.

Lemma 2.2.7 *G is a deduction graph if and only if the following hold*

1. *G is a finite closed box directed graph,*
2. *there is a box-topological ordering $<$ of G,*
3. *every node n of G is of one of the following four types:*

   A *n has no outgoing edges.*

   E *n has label B and has exactly two outgoing edges: one to a node $(m, A{\rightarrow}B)$ and one to a node (p, A), both within the scope of n.*

   I *n is a box-node of a box $\mathcal{B}$ with label $A{\rightarrow}B$ and has exactly one outgoing edge, which is to a node (j, B) inside the box $\mathcal{B}$ with no other ingoing edges. All free nodes of $\mathcal{B}$ have label A.*

   R *n has label A and has exactly one outgoing edge, which is to a node (m, A) that is within the scope of n.*

Proof

$\Rightarrow$ By induction on the construction of deduction graphs (Definition 2.2.2). For every construction case for deduction graphs we have to check the three properties stated in the Lemma. Properties (1) and (3) are immediate (note that for the $\rightarrow$-I case, property (1) is enforced by the definition). For property (2), we know from the induction hypothesis that there is a

box-topological ordering $<$ on smaller graphs. In the construction cases Axiom, Repeat, $\rightarrow$-I or $\rightarrow$-E, we make the new node that is introduced highest in the $<$-ordering, which yields a box-topological ordering on the new graph. In the construction case Join, we have two box topological orderings, $<_1$ on G_1 and $<_2$ on G_2. Then $G_1 \cup G_2$ can be given a box-topological ordering by taking the union of $<_1$ and $<_2$ and in addition putting $n < m$ for every $n \in G_1, m \in G_2$.

$\Leftarrow$ By induction on the number of nodes of G. Let $<$ be the box-topological ordering that is assumed to exist. Let n be the node that is maximal w.r.t. $<$. Then n must be on top level. When we remove node n, possibly including its box (if n is of type I), we obtain a graph G' that again satisfies the properties listed in the Lemma. By induction hypothesis we see that G' is a deduction graph. Now we can add node n again, using one of the construction cases for deduction graphs: Join if n is an A node, $\rightarrow$-E if n is an E node, $\rightarrow$-I if n is an I node Repeat if n is an R node.

The next definition gives two special construction of sub-graphs that we will use later. In general, a deduction graph has many "conclusions": top-level nodes without incoming edges. The first construction focuses on one node, which does not even have to be top-level, and omits all parts of the graph that are not reachable from this node. The resulting graph has necessarily just one "conclusion", just like a deduction in Gentzen-Prawitz style.

The second construction is a certain closure of the contents of a box.

Definition 2.2.8 *Let G be a deduction graph.*

1. *For n a node of G, we define the* subgraph of G generated from n, *G_n, as the subgraph of G consisting of all nodes reachable from n (with boxes inherited from G).*

2. *For $\mathcal{B}$ a box in G with discharged nodes (n_1, A), ..., (n_k, A), and (m, A) be a fresh node (i.e. not in $\mathcal{B}$), we define the m-*closure of $\mathcal{B}$, *$\mathcal{B}^m$, as the graph consisting of all nodes inside $\mathcal{B}$ plus all nodes that can be reached from within $\mathcal{B}$ in one step plus a new node (m, A) and arrows from (n_1, A), ..., (n_k, A) to m.*

We have the following as a simple corollary of Lemma 2.2.7.

Corollary 2.2.9

1. *If G is a deduction graph containing a node n, then G_n is also a deduction graph.*

2. *If G is a deduction graph containing a box $\mathcal{B}$ and m is a node not in $\mathcal{B}$, then $\mathcal{B}^m$ is also a deduction graph. (Note that in $\mathcal{B}^m$, all nodes that were not in $\mathcal{B}$ have become A-nodes.)*

Definition 2.2.10 *Let G be a deduction graph, let $\mathcal{B}$ be a box of G, and let φ be a box-topological ordering of G. We write $\varphi_{\mathcal{B}^m}$ for the restriction of φ to the top-level nodes of $\mathcal{B}^m$.*

2.2.2 Embeddings from other formalisms

Deduction graphs were meant to generalise both natural deductions (from Gentzen and Prawitz) and flag deductions of Fitch. It is not difficult to see that they do.

Definition 2.2.11 (Natural Deductions as Deduction Graphs) *Given a natural deduction Σ with conclusion B from assumptions Γ, we define a deduction graph $\bar{\Sigma}$ with a top-level node with label B and free nodes with labels Γ as follows.*

- *View all formula occurrences as a node and replace every part of the deduction of the form*

$$\frac{A \quad A{\rightarrow}B}{B} \qquad\qquad \frac{A}{A{\rightarrow}B}$$

  by edges from (m, B) to (n, A) and $(k, A{\rightarrow}B)$, and from $(m, A{\rightarrow}B)$ to (n, A) respectively. (Here m, n and k are the nodes associated with the specific occurrences of these formulas.)

- *In the $\rightarrow$-introduction rule, where $A{\rightarrow}B$ is concluded from B, discharging a number of assumptions A, let Σ be the natural deduction with conclusion B. We may assume (by induction) that we have a deduction graph $\bar{\Sigma}$ with top-level node (j, B). We now create a box $\mathcal{B}$, consisting of $\bar{\Sigma}$, and for all the top-level nodes (p, C) of $\bar{\Sigma}$ that correspond to occurrences of hypotheses C that are not discharged at the introduction of $A{\rightarrow}B$ we add a new node (p', C) outside $\mathcal{B}$ and a repeat edge from (p, C) to (p', C). This is indicated in Figure 2.7.*

Figure 2.7: From natural deductions to deduction graphs

The following is now immediate.

Lemma 2.2.12 *If Σ is a natural deduction with conclusion B from assumptions Γ, $\bar{\Sigma}$ is a deduction graph with a top-level node with label B and free nodes with labels Γ.*

Example 2.2.13 *Consider the following Gentzen-Prawitz style natural deduction:*

$$\dfrac{\dfrac{\dfrac{[A]^1 \quad A{\to}C}{C}}{B{\to}C} \qquad \dfrac{[A]^1 \quad A{\to}(B{\to}C){\to}D}{(B{\to}C){\to}D}}{\dfrac{D}{A{\to}D}\,1}$$

The translation of this in deduction graph is shown in Figure 2.8.

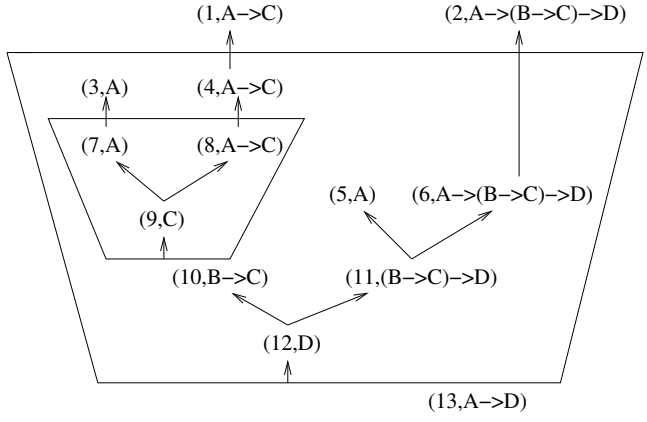
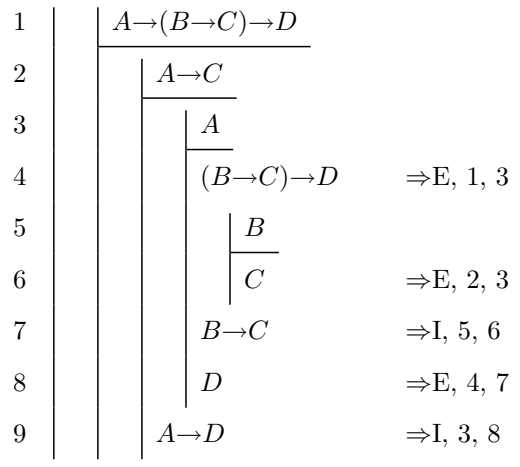


Figure 2.8: From natural deduction to deduction graph.

The rules for Fitch style flag deduction can be found in Appendix A. Here we restrict to a clarifying example, see Σ_1, the upper deduction of Figure 2.9. This deduction proves $A{\to}D$ (on the last line) from the hypotheses $A{\to}(B{\to}C){\to}D$ and $A{\to}C$, on lines 1 and 2. The open (i.e. undischarged) hypotheses are indicated by a 'flag' whose pole extends to the line of the conclusion. Discharging a hypothesis corresponds to ending the flag pole. On line 7, the flag pole of the hypothesis B ends and we conclude $B{\to}C$ from C, while ending the scope of the B flag. The comments on the right give the *motivation* for the conclusion on the line, referring to a logical operation and previous lines. The lines that can be referred to in a motivation should be *in scope*, a notion that should be intuitively clear. In Σ_3, the lower right Fitch deduction we see one additional rule, which is the *repeat* rule. This rule allows to repeat a formula that has been derived on a previous line, if it is in scope.

Definition 2.2.14 (Fitch Deductions as Deduction Graphs)
Given a Fitch deduction Σ with conclusion B and flags Γ, we define a deduction graph $\hat{\Sigma}$ with a top-level node with label B and free nodes with labels Γ as follows.

View a formula occurrence B on line n as a node (n, B) and then add edges as follows:



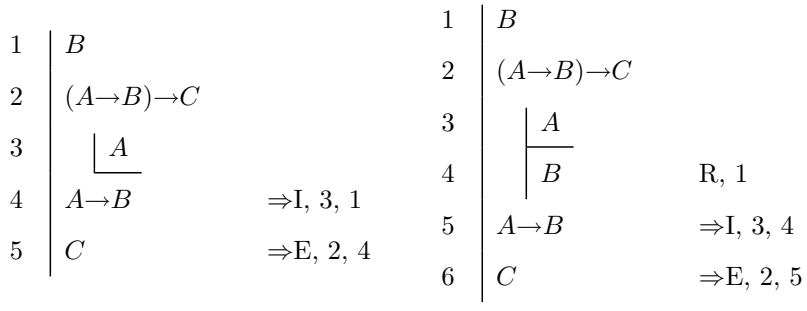


Figure 2.9: Fitch deductions Σ_1, Σ_2 and Σ_3

- *If $\to$-E, p, q is the motivation on line n, add edges from (n, B) to nodes p and q;*
- *If $\to$-I, p, q is the motivation on line n, add an edge from (n, B) to node q;*
- *If R, p is the motivation on line n, add an edge from (n, B) to node p;*
- *If there is a flag that starts at line i and is discharged at line $j + 1$, put a box around nodes $i, \ldots, j$.*

This creates a deduction graph, but there is one slight subtlety:

> *In flag deductions, if we introduce $A \to B$, B does not have to be on the line before the $\to$-introduction (See the lower left flag deduction Σ_2 in Figure 2.9). This means that:*
>
> 1. *B does not have to be under "flag" A;*

2. There might be more than one reference to the line of B.

Because in deduction graphs a box-node has an edge to a node in the box with no other incoming edges*, we first add a repeat step to the flag deduction, before performing the translation.*

The following lemma is now immediate, as can be seen from the examples in Figure 2.10, which are translations of the Fitch deductions of Figure 2.9.

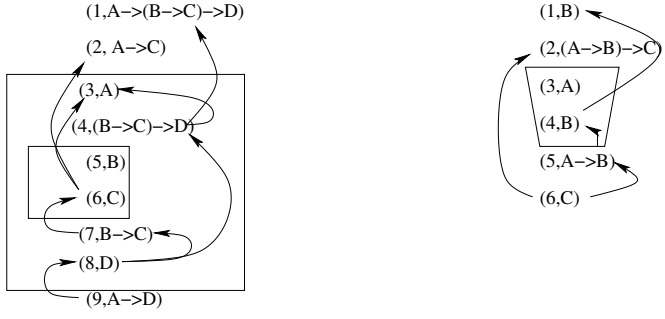


Figure 2.10: From flag deductions to deduction graphs

Lemma 2.2.15 *If Σ is a Fitch deduction with conclusion B and flags Γ, $\hat{\Sigma}$ is a deduction graph with a top-level node with label B and free nodes with labels Γ.*

Chapter 3

Cut-elimination

3.1 Introduction

In Gentzen-Prawitz natural deduction a *cut* is basically defined as a part of a deduction where the introduction of a connective (like $\rightarrow$) is immediately followed by elimination of the same connective. This is a "detour": It is not necessary in these cases to introduce the connectives at all. Prawitz [71] shows by transforming a deduction with a cut to a deduction without that cut, but with the same assumptions and conclusion, that all cuts can be eliminated. From Gentzen's result [27] one could also conclude this, but his proof is indirect.

The point of cut-elimination is *not* to make a deduction shorter. The transformations of Prawitz include copying of parts of a deduction, which in general makes a deduction even larger. No, the idea is to show that all theorems of propositional logic can be proven by a deduction of a special form: First the assumptions are broken down by elimination rules (the *analytic* part), and then the conclusion is composed by introduction rules (the *synthetic* part). This is not only a nice philosophical result, but it also implies that this part of logic is *decidable*.

In this chapter we study cuts and elimination of cuts for deduction graphs. Section 3.2 gives the definition of a cut in deduction graphs and discusses cut-free deduction graphs. By showing that a cut-free deduction graph also allows a division into an analytic part and a synthetic part, we argue that our definition makes sense. This is not very remarkable as our definition is very similar to Prawitz's: Only we need to consider repeats too.

Section 3.3 then discusses how to eliminate a cut from a deduction graph. Because a deduction graph is in general not a tree and because it contains boxes, this is much more involved than in Gentzen-Prawitz style natural deduction. To make it more surveyable, we divide the elimination of a cut in smaller transformations. Three of these smaller transformations rearrange boxes and handle repeats and sharing and hence they do not associate to cut-eliminating in Gentzen-Prawitz style natural deduction. The fourth transformation, the so-

called safe cut-elimination, is always the last in the process of eliminating a cut. This transformation comes close to cut-elimination in Gentzen-Prawitz natural deduction. But because deduction graphs allow sharing, safe cut-elimination does not copy parts of the deduction graph.

This chapter only shows how to eliminate one cut. In the process of eliminating a cut, we may create numerous new ones. So, it is a priori not clear, whether we can transform every deduction graph in a meaningful way into a cut-free deduction graph. That this is nevertheless possible will be discussed in the next chapter.

3.2 Cuts and Cut-free Deduction Graphs

We now define the notion of a cut in deduction graphs. It is basically an $\rightarrow$-I followed by an $\rightarrow$-E, although these two might be separated by several repeats (see Figure 3.1). Note that the definition doesn't say anything about the (relative) depths of the nodes involved. Nor does it speak about any possible other edges to them.

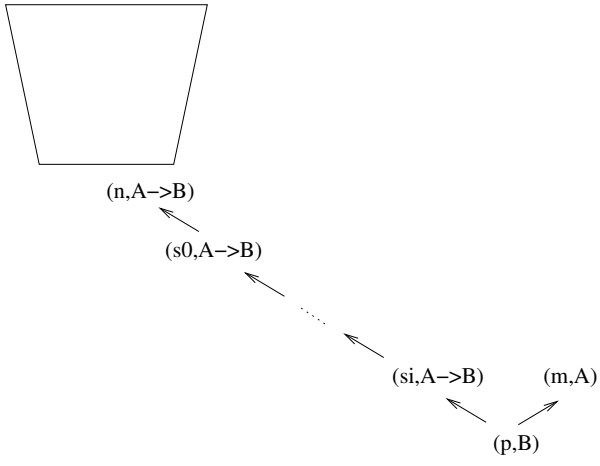


Figure 3.1: A cut in a deduction graph.

Definition 3.2.1 (Cut) *A* cut *in a deduction graph G is a subgraph of G consisting of*

- *A box-node* $(n, A{\rightarrow}B)$,
- *A node* (p, B),
- *A node* (m, A),
- *A sequence of* R*-nodes* $(s_0, A{\rightarrow}B), \ldots, (s_i, A{\rightarrow}B)$,

- *Edges* $(p, B) \longrightarrow\!\triangleright (s_i, A{\rightarrow}B) \longrightarrow\!\triangleright \ldots \longrightarrow\!\triangleright (s_0, A{\rightarrow}B) \longrightarrow\!\triangleright (n, A{\rightarrow}B)$,
- *An edge* $(p, B) \longrightarrow\!\triangleright (m, A)$.

Definition 3.2.2 (Cut-free deduction graph) *A deduction graph is called* cut-free*, when none of its subgraphs is a cut.*

Definition 3.2.3 (Major premiss) *Let G be a deduction graph and let (n, B) be an* E*-node of G with edges $(n, B) \longrightarrow\!\triangleright (k, A{\rightarrow}B)$ and $(n, B) \longrightarrow\!\triangleright (l, A)$. Then $(k, A{\rightarrow}B)$ is called the* major premiss *of (n, B).*

Definition 3.2.4 (Minor premiss) *Let G be a deduction graph and let (n, B) be an* E*-node of G with edges $(n, B) \longrightarrow\!\triangleright (k, A{\rightarrow}B)$ and $(n, B) \longrightarrow\!\triangleright (l, A)$. Then (l, A) is called the* minor premiss *of (n, B).*

Sometimes we will also speak of *the box-node of a cut*, *the* I*-node of a cut*, *the major premiss of a cut*, *the minor premiss of a cut*, and the E*-node of a cut*.

Note that, in contrast to Gentzen-Prawitz style natural deduction, a node can be both a major premiss and a minor premiss. It is, however, never the major premiss and the minor premiss of the same node.

Definition 3.2.5 (Branch) *A* branch *of a deduction graph consists of:*

1. *A free node n_0;*
2. *A node n_k that is either without incoming edges, or that is a minor premiss;*
3. *Nodes $n_1, \ldots, n_{k-1}$, such that n_i is not the minor premiss of n_{i+1} for $1 \leq i < k$;*
4. *Edges $n_k \longrightarrow\!\triangleright n_{k-1} \longrightarrow\!\triangleright \ldots \longrightarrow\!\triangleright n_1 \longrightarrow\!\triangleright n_0$.*

Prawitz divides every branch in natural deduction in three parts (analytic, minimal and synthetic) in a unique way. Because deduction graphs allow the repetition of formulas, we loose the uniqueness.

Theorem 3.2.6 *A branch of a cut-free deduction graph can be divided into three parts:*

1. *An* analytic *part $n_0, \ldots, n_{l-1}$, in which each node is the major premiss of the elimination that constructs the succeeding node in the sequence, or the succeeding node in the sequence in an* R*-node.*
2. *A* minimal node n_l.
3. *A* synthetic *part $n_{l+1}, \ldots, n_k$, in which each node is an* I*-node or an* R*-node.*

Proof Let G be a cut-free deduction graph. Suppose $n_k \rightarrowtriangle n_{k-1} \rightarrowtriangle \ldots \rightarrowtriangle n_1 \rightarrowtriangle n_0$ are the nodes and edges of a branch of G. Because G does not contain cuts, it cannot be the case that n_i is an I-node, n_j is a major premiss and $i < j$. So, all major premisses come first (possibly interlarded with premisses of R nodes), followed by all I-nodes (possibly interlarded with R-nodes).

Theorem 3.2.6 shows that the notion of cut in deduction graphs (Definition 3.2.1) makes sense.

3.3 Eliminating a Cut

3.3.1 Safe cut-elimination

We want to eliminate a cut in a deduction graph very similar to how it is done in Gentzen-Prawitz natural deduction, but this time by rearranging the nodes, edges and boxes. This rearrangement may also include adding and deleting nodes, edges and boxes. If we put aside the possible R-nodes between the I-node and the E-node, this basically amount to Figure 3.2. The multiple edges to (m, A) replace the copying that is involved in the transformation in Gentzen-Prawitz style natural deduction.

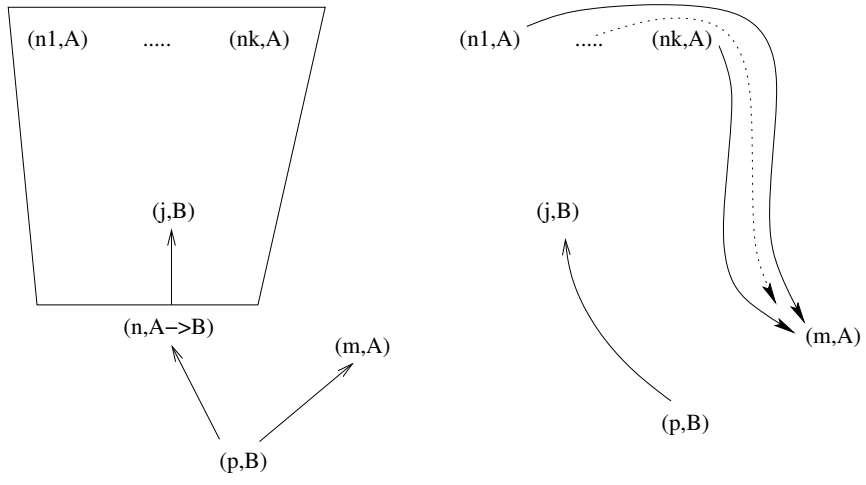


Figure 3.2: Cut-elimination in deduction graphs.

The actual situation, however, is harder than the one sketched in Figure 3.2. The addition of a repeat step, the graph structure and the box structure lead to the following three difficulties respectively:

1. The I-node and the E-node may be separated by R-nodes.

2. The involved box may have multiple incoming edges. In this case, if we would transform the deduction graph as done in the figure, the resulting structure would not be a deduction graph again, as these edges would be "dangling".

3. The minor premiss of the elimination might be at a greater depth than the major premiss. In this case, if we would transform the deduction graph as done in the figure, the resulting graph may not be a closed box directed graph, as edges from a node $((n_1, A), \ldots, (n_k, A)$ in the figure) might go to a node at a greater depth $((m, A))$.

How to handle these cases in discussed in the Sections 3.3.2, 3.3.3 and 3.3.4. Here we discuss the elimination of a cut, in which these difficulties do not appear, called a *safe cut*.

Definition 3.3.1 *A cut in a graph G is* safe *if the following requirements hold:*

- *there is an edge from (p, B) to $(n, A{\rightarrow}B)$ and that is the only edge to $(n, A{\rightarrow}B)$;*
- *the nodes $(n, A{\rightarrow}B)$ and (p, B) are at the same depth.*

The requirement on the depths of the nodes in Definition 3.3.1 is stronger than seems to be necessary: We require that the major premiss and the E-node are at the same depth, instead of the major premiss and the minor premiss being at the same depth. We decide on the stronger requirement, because it turns out to be advantageous, both for the translation to λ-calculus (Chapter 4) and for modularity when deduction graphs are extended with a "for all"-quantification (Chapter 7).

Definition 3.3.2 eliminating a safe cut *is the following transformation of a deduction graph.*

- *remove the edges to and from n,*
- *remove the edge from (p, B) to (m, A),*
- *remove the box node n*
- *add an edge from (p, B) to (j, B) (the node that n pointed to),*
- *add edges from $n_1, \ldots, n_k$ (the free nodes of the box) to (m, A).*

Example 3.3.3 *In the deduction graph of Example 2.2.4, there is a cut that we can eliminate. If we do so we obtain the following deduction graph.*

Lemma 3.3.4 *If G is a deduction graph with safe cut c and G' is obtained from G by eliminating c, then G' is also a deduction graph.*

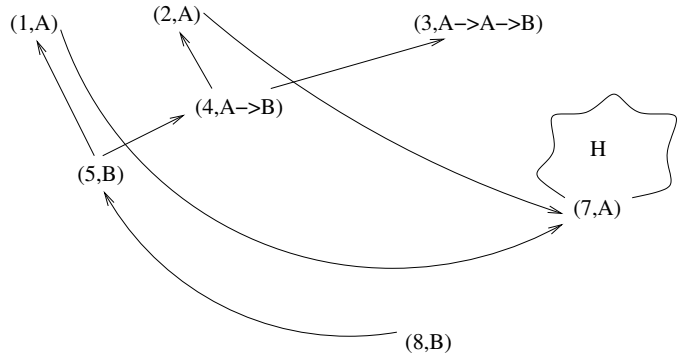


Figure 3.3: Deduction graph of Figure 2.6 after a cut-elimination step.

Proof We use Lemma 2.2.7. All nodes in G' are of the right form: A, E, I or R. To see that G' is a closed box directed graph, we verify the property that there are no edges pointing into a box: (m, A) is in scope of $(n, A{\to}B)$, so (also after removing the box node), (m, A) is in scope of $n_1, \ldots, n_k$ and the edges from $n_1, \ldots, n_k$ to m do not point into a box. Finally, to find a topological order on G', note that there is no edge from m to n and no edge from m to any node inside the eliminated box $\mathcal{B}$. So, without loss of generality, we may assume that $l > m$ for all $l \in \mathcal{B}$.

3.3.2 Repeat-elimination

If a cut is not safe, we first want to make it safe before performing a cut-elimination step. A first problem with non-safe deduction graphs is that the $\to$-I step and the $\to$-E step could be separated by Repeats. See Figure 3.4 for an example. To eliminate the cut that arises from $A{\to}B$ and A, we first have to eliminate the Repeats.

Definition 3.3.5 (Cut hidden by repeats) *Let G be a deduction graph that contains a cut c. If the* I*-node of c and the* E*-node of c are separated by* R*-nodes, we call c a* cut hidden by repeats.

Definition 3.3.6 (Repeat-elimination) *Let G be a deduction graph that contains it contains a node $(n_0, A{\to}B)$, an* R*-node $(n_1, A{\to}B)$, a node (n_2, B), a node (n_3, A) and edges $n_1 \twoheadrightarrow n_0$, $n_2 \twoheadrightarrow n_1$ and $n_2 \twoheadrightarrow n_3$. The* repeat-elimination *at n_0, n_1, n_2 is obtained by:*

- *When an edge points to n_1, redirect it to n_0;*
- *Remove n_1.*

Figure 3.5 shows the deduction graph of Figure 3.4 after repeat-elimination.

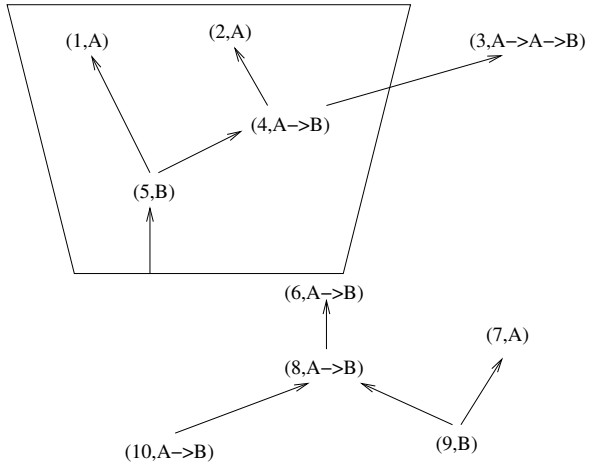


Figure 3.4: Deduction graph with a cut hidden by repeats.

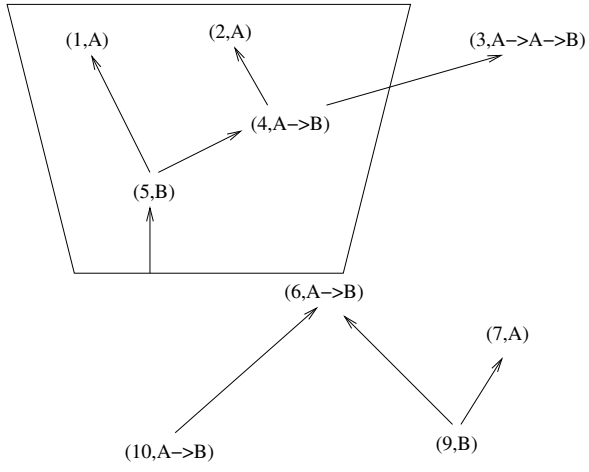


Figure 3.5: The deduction graph of Figure 3.4 after repeat-elimination.

Lemma 3.3.7 *Let G be a deduction graph with a cut hidden by repeats. Let G' be the graph after repeat-elimination. Then G' is also a deduction graph.*

Proof We have to check that G' is again a closed box directed graph. The only requirement that is not directly obvious is that box-nodes have an outgoing edge that points to a node inside the box. G' answers this requirement, because the node a box-node points to, does not have other incoming edges. Figure 3.6 shows what would go wrong otherwise. That G' answers the other requirements of deduction graphs too, is easy.

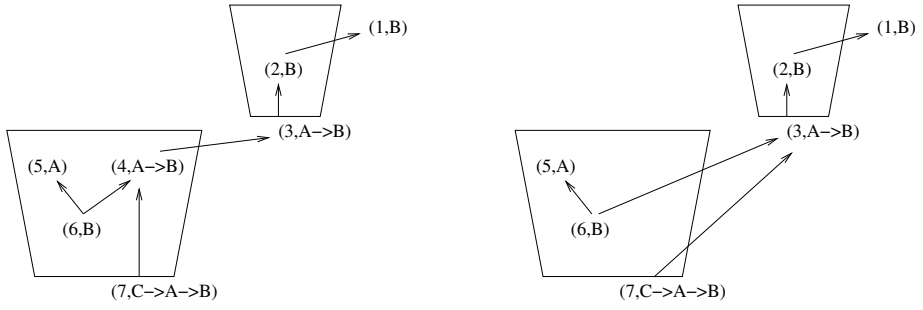


Figure 3.6: Incorrect deduction graph (left) and the graph after repeat-elimination.

Remark that if G contains a cut hidden by repeats, we can do a repeat-elimination.

3.3.3 Unsharing

Another problem arises when a boxed part is "shared", see Figure 3.7.

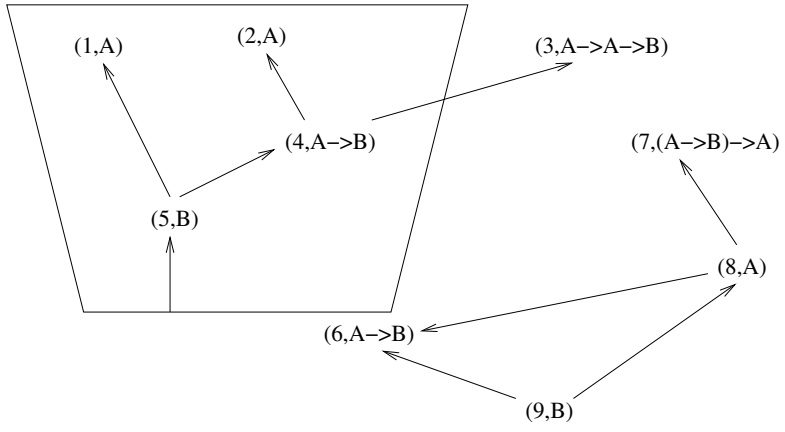


Figure 3.7: Deduction graph with a cut hidden by sharing.

Definition 3.3.8 (Cut hidden by sharing) *Let G be a deduction graph that contains a cut c. If the* |*-node of c has 2 or more incoming edges, then we call c a* cut hidden by sharing.

Definition 3.3.9 (Unsharing) *Let G be a deduction graph that contains a box $\mathcal{B}$ with box-node $(n, A{\rightarrow}C)$ and $k \geq 2$ ingoing edges, from $p_1, \ldots, p_k$. Then the* unsharing of G at nodes $n, p_1, \ldots, p_k$ *is obtained by (see Figure 3.8)*

- *making a box $\mathcal{B}'$, that contains a copy of all nodes and edges of $\mathcal{B}$,*
- *copy all the outgoing edges of $\mathcal{B}$ to $\mathcal{B}'$ (thus if we had $q \longrightarrow\!\triangleright\, m$ with $q \in \mathcal{B}$, $q' \in \mathcal{B}'$ and $m \notin \mathcal{B}$, then we add $q' \longrightarrow\!\triangleright\, m$, where q' is the copy of q in $\mathcal{B}'$),*
- *letting $p_2, \ldots p_k$ point to n' (the box-node of $\mathcal{B}'$) instead of n.*

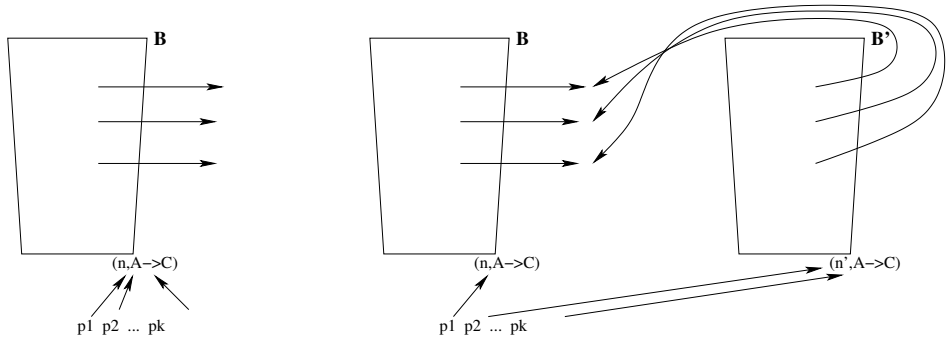


Figure 3.8: The idea of unsharing.

Figure 3.9 shows the unsharing of the deduction graph of Figure 3.7.

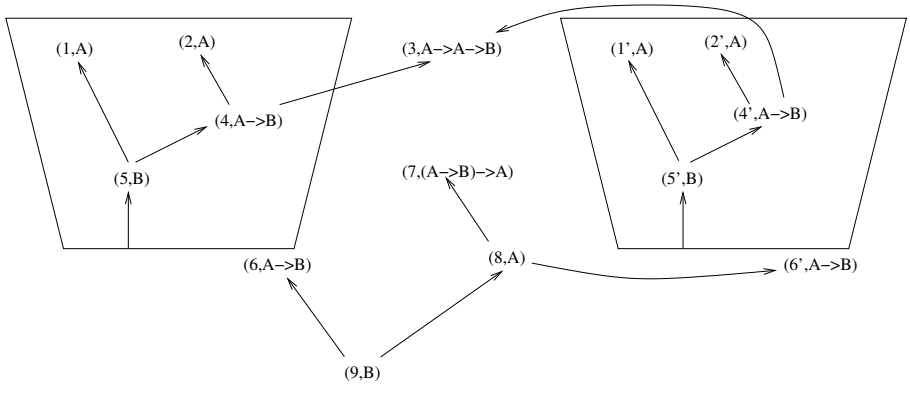


Figure 3.9: The deduction graph of 3.7 after unsharing.

Lemma 3.3.10 *Let G be a deduction graph that contains a box $\mathcal{B}$ with box-node n with label $A \rightarrow B$ and $k \geq 2$ ingoing edges, from $p_1, \ldots p_k$. Then the unsharing of G at nodes $n, p_1, \ldots, p_k$ is a deduction graph.*

Proof We use Lemma 2.2.7. The graph G' is a finite closed box directed graph. We make a box-topological ordering of G' by putting all nodes of $\mathcal{B}'$ immediately

after the last node of $\mathcal{B}$ in the box-topological order of G. Finally, each node of G' satisfies one of the cases of Lemma 2.2.7.

Remark that if G is a deduction graph with a cut hidden by sharing, we can do an unsharing.

3.3.4 Incorporation

In Figure 3.10 three possible positions of the cut with respect to boxes are depicted. In the third case, contracting the cut gives us an edge pointing into a box, so we cannot contract it. As explained in Section 3.2, we do not allow contraction in the second case either. In these cases we first do *incorporation*, which puts the box with box-node n in all boxes that p is in, level by level.

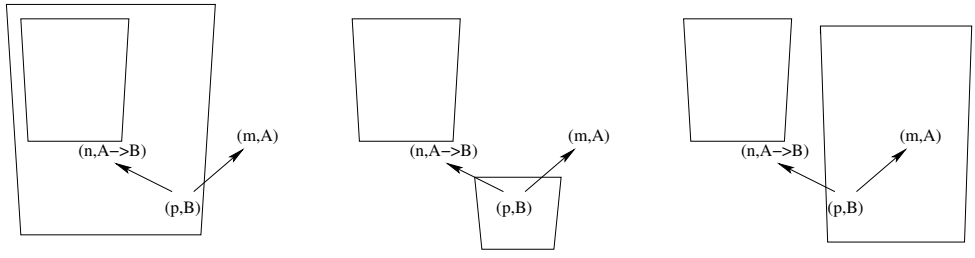


Figure 3.10: Three possible positions of the cut w.r.t. boxes. When n and p are at the same depth (first picture) the cut is *safe*. In the other cases we have a *depth-conflict*.

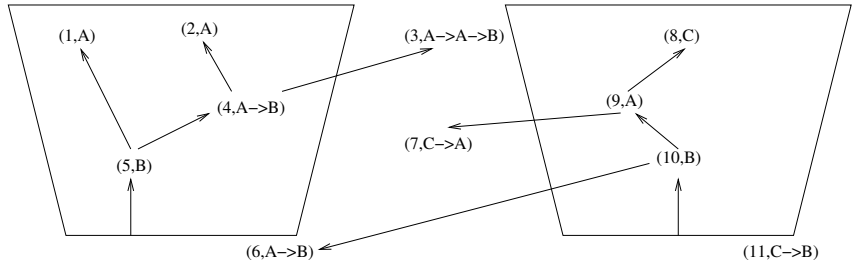


Figure 3.11: Deduction graph with hidden cut due to a depth conflict

Definition 3.3.11 (Cut hidden by depth conflict) *Let G be a deduction graph with a cut c. If the major premiss of c is at a greater depth than the* E*-node, we call c a cut hidden by depth conflict.*

See Figure 3.11 for an example of a depth conflict.

Definition 3.3.12 (Incorporation) *Let G be a deduction graph containing a box $\mathcal{B}$ with box-node $(n, A{\rightarrow}B)$ that has exactly one ingoing edge, which is from a node p, and p is at a greater depth than $(n, A{\rightarrow}B)$. In that case the* incorporation of G at n, p *is obtained by moving box $\mathcal{B}$ into the largest box that excludes $(n, A{\rightarrow}B)$, but includes p.*

Figure 3.12 shows the incorporation of the deduction graph of Figure 3.11.

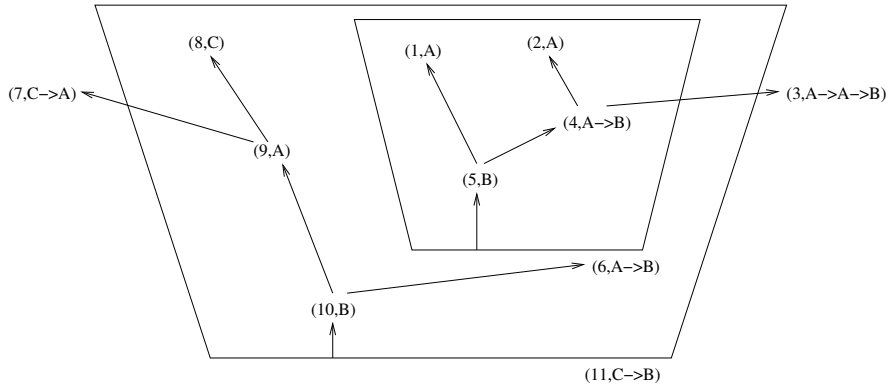


Figure 3.12: Deduction graph of Figure 3.11 after incorporation.

Lemma 3.3.13 *Let G be a deduction graph containing a box $\mathcal{B}$ with box-node $(n, A{\rightarrow}B)$ that has exactly one ingoing edge, which is from a node p, and p is at a greater depth than $(n, A{\rightarrow}B)$. Then the incorporation of G at n, p is a deduction graph.*

Proof Because no other edges point to n, the resulting graph is a finite closed box directed graph. The nodes in the resulting graph are all of the same type as they were in G. Also the topological order can be taken the same as for G.

Remark that if G is a deduction graph with a cut c hidden by depth conflict, and c is not hidden by repeats, nor is it hidden by sharing, then we can do an incorporation.

3.3.5 The process of eliminating a cut

To eliminate a cut in a deduction graph in general, we apply some of the transformations that are defined in the previous sections in succession. This amounts to the so-called *process of cut-elimination.*

Definition 3.3.14 *Given a deduction graph G with a cut c,* the process of eliminating the cut c *is the following.*

1. *As long as there is no edge from (p, B) to $(n, A{\rightarrow}B)$, perform the appropriate* **repeat-elimination** *step as described in Definition 3.3.6;*

2. *If there is an edge from* (p, B) *to* $(n, A{\rightarrow}B)$ *and this is not the only edge to* $(n, A{\rightarrow}B)$*, perform the appropriate* **unsharing** *step, as defined in Definition 3.3.9;*

3. *As long as* (p, B) *is not at the same depth as* $(n, A{\rightarrow}B)$ *perform an* **incorporation** *step, as described in Definition 3.3.11;*

4. *If* c *is safe, perform the* **safe cut-elimination** *step as defined in Definition 3.3.1.*

Chapter 4

Computational Content

4.1 Introduction

The λ-calculus has been introduced by Church [17] and Kleene [48] to describe formally the class of computable functions and to provide foundations for logic and mathematics. It has turned out later (see Chapter 1) that the system of *typed* λ-calculus is isomorphic to the system of natural deduction proofs. This isomorphism is called the *Curry-Howard-de Bruijn isomorphism*. The fact that there are as many typed computable functions as natural deduction proofs is not surprising at all: most formal systems describe countably many objects. It is quite amazing, however, that β-reduction of λ-terms reflects cut-elimination in natural deduction.

Section 4.2 gives a translation $[\![-]\!]$ from deduction graphs to simply typed λ-terms. This translation is very straight-forward: Basically, it unshares a deduction graph G seen from a node n to a natural deduction in tree form, and computes the λ-term associated to the latter. Therefore it is not remarkable that the transformations repeat-elimination, incorporation and unsharing are not reflected in the β-reduction of the term $[\![G, n]\!]$. The merits of the translation are that it provides a *Soundness Theorem* for deduction graphs. When we change the translation such that it maps to a *non-erasing* calculus, we get another desirable result: *Strong Normalisation* for the process of cut-elimination on deduction graphs.

Section 4.3 studies a context-calculus with let-bindings. Contexts are terms with a "hole". Let-bindings provide some kind of sharing, which by changing the notation a little bit, can also be seen as explicit substitutions. There are many well-known let-calculi and explicit substitution-calculi [64] [59] [13] [46], but the one we present is significantly different. We discuss that in Section 4.6.

Deduction graphs do not have just one conclusion in general; they may have several. This makes that a translation to terms will always be a bit off, as a term only has one type. The translation $\langle\!\langle - \rangle\!\rangle$ of Section 4.4 maps deduction graphs to a context-calculus with let-bindings. Filling the hole of a context then reflects

focussing on one of the deduction graph's conclusions. The let-binding, as said before, takes care of sharing, but it also offers an easy way to mirror repeats. It is then shown that the translation reflects the process of cut-elimination on deduction graphs. Note that we have added extra elements that correspond to the additional step and structure of the deduction graph, but non of our additions correspond to the notion of a box. The box is already present as the scope of a λ-binding in simply typed λ-calculus.

In Section 4.5 we connect the translation to simply typed λ-terms $[\![-]\!]$ and the translation to contexts with let-bindings $\langle\!\langle - \rangle\!\rangle$, by filling the holes and getting rid of the lets.

4.2 Simply typed λ-calculus

4.2.1 Translation to simply typed λ-terms

We now map deduction graphs to simply typed λ-terms. Every node in a deduction graph can be mapped to a term, the term corresponding to the proof of the formula in that node.

Definition 4.2.1 *Given a deduction graph G and a node n in G, we define the λ-term $[\![G, n]\!]$ as follows (by induction on the number of nodes of G).*

- A *If (n, A) has no outgoing edges, $[\![G, n]\!] := x^A_n$,*
- E *If $(n, B) \twoheadrightarrow (m, A \to B)$, and $(n, B) \twoheadrightarrow (p, A)$, define $[\![G, n]\!] := [\![G_m, m]\!]\,[\![G_p, p]\!]$.*
- I *If $(n, A \to B)$ is a box-node with $(n, A \to B) \longrightarrow\!\!\triangleright\, (j, B)$ and the free nodes of the box are $(n_1, A), \ldots, (n_k, A)$, define $[\![G, n]\!] := \lambda x{:}A.([\![G_j, j]\!][x_{n_1} := x, \ldots, x_{n_k} := x])$.*
- R *If $(n, A) \longrightarrow\!\!\triangleright\, (m, A)$, define $[\![G, n]\!] := [\![G_m, m]\!]$*

Remark that $[\![G, n]\!] = [\![G_n, n]\!]$ for all n.

Lemma 4.2.2 *Let G be a deduction graph and let (n, A) be a node of G. Then $[\![G, n]\!]$ is of type A.*

Proof By induction.

Theorem 4.2.3 (Soundness) *Let G be a deduction graph and let (n, B) be a node of G. Let $(n_1, A_1), (n_2, A_2), \ldots, (n_k, A_k)$ be the free nodes of G_n. Then there exists a Gentzen-Prawitz style natural deduction of B from $A_1, A_2, \ldots A_k$.*

Proof By the Curry-Howard-de Bruijn isomorphism (see Chapter 2) and Lemma 4.2.2.

This mapping ignores quite a lot of structure of the graph. The following lemma is an immediate consequence of the definitions.

Lemma 4.2.4 *Let G, G' be a deduction graph and let n be a node of G and G'. Then:*

- *If G' is obtained from G by an unsharing step, then $[\![G, n]\!] = [\![G', n]\!]$.*
- *If G' is obtained from G by an incorporation step, then $[\![G, n]\!] = [\![G', n]\!]$.*
- *If G' is obtained from G by an* R*-elimination step, then $[\![G, n]\!] = [\![G', n]\!]$.*

We now connect the process of cut-elimination on deduction graphs with β-reduction on simply typed λ-terms. We first relate substitution in λ-terms to a notion of substitution in deduction graphs. In deduction graphs, substitution is performed by turning an A-node into an R-node, by linking it to a node with the same label.

Definition 4.2.5 *Let G and G' be two deduction graphs and suppose that m_0, …, m_k are the free nodes of G. The function f from the nodes of G to the nodes of G' is a* deduction graph substitution *(or* dg-substitution*) if the following hold for all nodes m, n_0, n_1 of G.*

1. *the label of m is the label of $f(m)$;*
2. *if $n_0 \longrightarrow\!\triangleright\, n_1$, then $f(n_0) \longrightarrow\!\triangleright\, f(n_1)$;*
3. *$m \in \mathcal{B}$ iff $f(m) \in f(\mathcal{B})$;*
4. *If $m \neq m_0, \ldots, m_k$ is an* A/E/I/R *node, then $f(m)$ is an* A/E/I/R *node too.*

In the third clause of the definition, we use $f(\mathcal{B})$. This clause should obviously be understood as '$m \in \mathcal{B}$ where $\mathcal{B}$ has box node n iff $f(m) \in \mathcal{B}'$ where $\mathcal{B}'$ has box node $f(n)$'.

A dg-substitution cannot change much of the graph: G' may contain parts that G doesn't have and G' may have some more sharing than G. But most importantly, a top level A-node of G may be 'replaced by' some other node.

Example 4.2.6

- *For every deduction graph G, the identity on G is a dg-substitution.*
- *Suppose G and G' are deduction graphs and G' is obtained from G by an unsharing. Then the obvious "sharing" map from G' to G is a dg-substitution.*
- *Consider the deduction graphs of Figure 4.1. The map $i \mapsto i$ is a dg-substitution from the left one to the right one.*

The correspondence with substitution is stated in the following lemma.

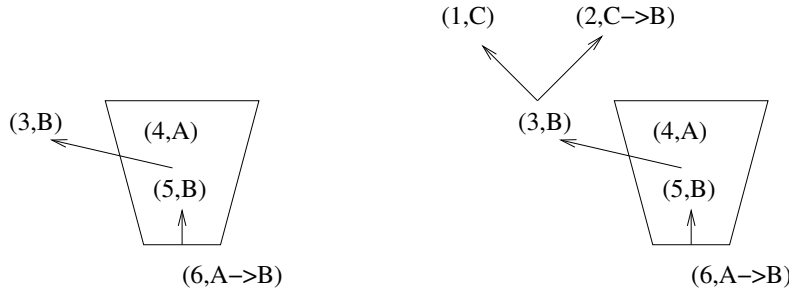


Figure 4.1: A deduction graph substitution.

Lemma 4.2.7 *Let G and G' be two deduction graphs with $m_0, \ldots, m_k$ the free nodes of G and n a top-level node of G. Let f be a dg-substitution from G to G'. Then the following holds.*

$$[\![G', f(n)]\!] = [\![G, n]\!][\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]]$$

where $[\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]]$ is an abbreviation for $[x_{m_0} := [\![G', f(m_0)]\!], \ldots, x_{m_k} := [\![G', f(m_k)]\!]]$.

Proof The proof is by induction on the number of reachable nodes from n. By $\overset{\text{IH}}{=}$ we indicate an application of the induction hypothesis.

A If (n, A) has no outgoing edges, then there is an i such that $n = m_i$.

$$\begin{array}{rcl} [\![G', f(m_i)]\!] & = & x_{m_i}[x_{m_i} := [\![G', f(m_i)]\!]] \\ & = & [\![G, m_i]\!][\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]]. \end{array}$$

E If $(n, B) \longrightarrow\!\triangleright (q, A{\to}B)$, and $(n, B) \longrightarrow\!\triangleright (p, A)$, then

$$\begin{array}{rcl} [\![G', f(n)]\!] & = & [\![G', f(q)]\!][\![G', f(p)]\!] \\ & \overset{\text{IH}}{=} & ([\![G', q]\!][\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]])([\![G, p]\!][\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]]) \\ & = & ([\![G, q]\!][\![G, p]\!])[\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]] \\ & = & [\![G, n]\!][\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]] \end{array}$$

I If $(n, A{\to}B)$ is a box-node with $(n, A{\to}B) \longrightarrow\!\triangleright (j, B)$ and the free nodes of the box are $(n_1, A), \ldots, (n_l, A)$, then $f(n_1), \ldots, f(n_l)$ are free nodes of the box with box-node $f(n)$. We use $[\overrightarrow{x_n} := x]$ as abbreviation for $[x_{n_1} := x, \ldots, x_{n_l} := x]$, we use $[\overrightarrow{x_{f(n)}} := x]$ as abbreviation for $[x_{f(n_1)} := x, \ldots, x_{f(n_l)} := x]$ and we use $[\overrightarrow{x_n} := [\![G', \overrightarrow{f(n)}]\!]]$ as abbreviation for $[x_{n_1} := [\![G', f(n_1)]\!], \ldots, x_{n_l} := [\![G', f(n_l)]\!]]$.

$$[\![G', f(n)]\!] \quad = \quad \lambda x{:}A.([\![G, f(j)]\!][\overrightarrow{x_{f(n)}} := x])$$

$$\begin{array}{rcl} & \overset{\text{IH}}{=} & \lambda x{:}A.(([\![G,j]\!][\overrightarrow{x_n} := [\![G', \overrightarrow{f(n)}]\!]][\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]]) \\ & & \qquad [\overrightarrow{x_{f(n)}} := x]) \\ & \overset{*}{=} & \lambda x{:}A.(([\![G,j]\!][\overrightarrow{x_n} := x])[\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]]) \\ & = & (\lambda x{:}A.([\![G,j]\!][\overrightarrow{x_n} := x]))[\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]] \\ & = & [\![G,n]\!][\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]] \end{array}$$

The equality $\overset{*}{=}$ uses the fact that $\overrightarrow{m}$ and $\overrightarrow{n}$ are disjoint and that $[\![G', f(n_i)]\!] = x_{f(n_i)}$ (because n_i is an A node inside a box and so $f(n_i)$ is an A node too).

R If $(n, A) \longrightarrow\!\!\triangleright (q, A)$, then

$$\begin{array}{rcl} [\![G', f(n)]\!] & = & [\![G', f(q)]\!] \\ & \overset{\text{IH}}{=} & [\![G, q]\!][\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]] \\ & = & [\![G, n]\!][\overrightarrow{x_m} := [\![G', \overrightarrow{f(m)}]\!]]. \end{array}$$

Lemma 4.2.7 is used in the proof of the following lemma:

Lemma 4.2.8 (Cut-elimination is β-reduction) *If $G \to_{\text{cut}} G'$ and r is a node of both G and G' such that $G_r \to_{\text{cut}} G'_r$, then:*

$$[\![G, r]\!] \twoheadrightarrow_\beta [\![G', r]\!].$$

Proof Due to Lemma 4.2.4 we only have to consider the case that the cut of G is safe. See Figure 4.2. We first consider the case that $r = (p, B)$. Let $M := [\![G, j]\!]$ and $N := [\![G, m]\!]$. Then $[\![G, n]\!] = (\lambda x{:}A.(M[x_{n_1} := x, \ldots, x_{n_k} := x])N)$. Let F be the deduction graph obtained from G by:

- removing the edges to and from n,
- removing the edge from (p, B) to (n, A),
- removing the box-node n,
- adding an edge from (p, B) to (j, B).

Then F is a deduction graph and $[\![F, r]\!] = [\![F, p]\!] = [\![G, j]\!] = M$. The identity is a deduction graph substitution from F to G', so

$$\begin{array}{rcl} [\![G', r]\!] & = & [\![F, r]\!][x_{n_1} := N, \ldots, x_{n_k} := N] \\ & = & M[x_{n_1} := N, \ldots, x_{n_k} := N] \end{array}$$

So $[\![G, r]\!] \to_\beta [\![G', r]\!]$. The case that $r \neq (p, B)$ is proven as follows. Note that p is at the top-level of G_p. And: G_r reduces to G'_r in one cut-elimination step. Because of the previous case and the Lemma 4.2.7, we get $[\![G, r]\!] \twoheadrightarrow_\beta [\![G', r]\!]$.

The lemma can be made more precise: if there is a path from r to the cut-formula, then we know that the β-reduction is not empty. If the cut-formula is not reachable from r, the reduction is a zero-step reduction and the right-hand side and left-hand side are the same.

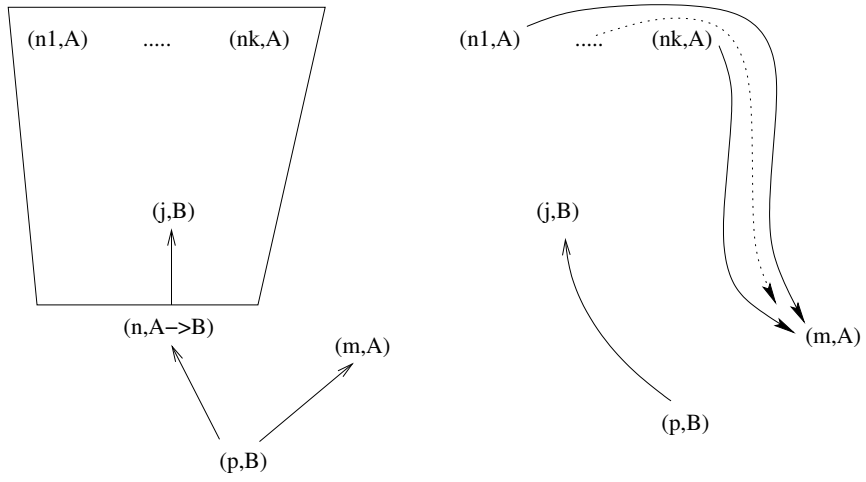


Figure 4.2: Cut-elimination in deduction graphs.

4.2.2 Strong normalisation of cut-elimination

As the reduction in Lemma 4.2.8 may be empty, we cannot conclude strong normalisation (SN) for cut-elimination from strong normalisation for β. A seemingly simple way to solve this is to consider the set $[\![G, n_1]\!], \ldots, [\![G, n_p]\!]$ for all top level nodes without incoming edges $n_1, \ldots, n_p$, and to prove that if $G \twoheadrightarrow_{\mathrm{cut}} G'$, then one of the terms $[\![G, n_i]\!]$ does a real reduction step. (So then the sum of the lengths of the maximal reduction sequences from $[\![G, n_1]\!], \ldots, [\![G, n_p]\!]$ may be used as a measure.) The problem with this reasoning is twofold:

1. If we do a cut-elimination where the $\rightarrow$-I is empty (there are no $n_1, \ldots, n_k$), we create a new top-level node (the (m, A) that we 'cut' with) without incoming edges.

2. A box may contain nodes without ingoing edges. These become (new) top-level nodes after a cut-elimination step.

Problem (1) cannot just be solved by taking $[\![G, n]\!]$ for all nodes n, because in the unsharing phase, nodes may get copied. Problem (2) cannot be solved by taking $[\![G, n]\!]$ for all nodes without incoming edges (including the ones inside boxes), because if the box gets removed – due to a cut-elimination step – we have to substitute inside $[\![G, n]\!]$.

The solution is to collect in one "term" all the λ-terms that arise from a node without incoming edges. To do that we extend the simply typed λ-calculus with a new term construction, *tupling*, $\langle -, \ldots \rangle$. So we have the following new term syntax and additional derivation rule.

$$\mathsf{T}_{\langle\rangle} ::= x \mid (\mathsf{T}_{\langle\rangle}\mathsf{T}_{\langle\rangle}) \mid \lambda x : A.\mathsf{T}_{\langle\rangle} \mid \langle \mathsf{T}_{\langle\rangle}, \mathsf{T}_{\langle\rangle}, \ldots, \mathsf{T}_{\langle\rangle} \rangle$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N_1 : B_1 \quad \ldots \quad \Gamma \vdash N_k : B_k}{\Gamma \vdash \langle M, N_1, \ldots, N_k \rangle : A}$$

Definition 4.2.9 *We define $\lambda{\to}\langle\rangle$ as the λ-calculus with the tupling constructor $\langle -, \ldots \rangle$ and the above derivation rule. The reduction rules for $\lambda{\to}\langle\rangle$ are as follows.*

$$\begin{array}{rcl}
(\lambda x{:}A.M)N & \longrightarrow_{\bar\beta} & M[x := N] \text{ if } x \in FV(M) \\
(\lambda x{:}A.M)N & \longrightarrow_{\bar\beta} & \langle M, N \rangle \text{ if } x \notin FV(M) \\
\langle M, P_1, \ldots, P_k \rangle N & \longrightarrow_{\bar\beta} & \langle MN, P_1, \ldots, P_k \rangle \\
\langle \ldots, \langle M, P_1, \ldots, P_k \rangle, \ldots \rangle & \longrightarrow_{\bar\beta} & \langle \ldots, M, P_1, \ldots, P_k, \ldots \rangle \\
N\langle M, P_1, \ldots, P_k \rangle & \longrightarrow_{\bar\beta} & \langle NM, P_1, \ldots, P_k \rangle
\end{array}$$

As can be observed from the rules, the $N_1, \ldots, N_k$ in $\langle M, N_1, \ldots, N_k \rangle$ act as a kind of 'garbage'. The order of the terms in $N_1, \ldots, N_k$ is irrelevant and we therefore consider terms *modulo* permutation of these vectors, which we will write as $\equiv_p$.

The ideas that we use in this section are very reminiscent of the work of Klop [49], where techniques are proposed (and used) to prove strong normalisation from weak normalisation (WN). This goes back to older work of Nederpelt [66], proving SN for Automath systems via a proof of WN. A good overview for the work of Klop and Nederpelt, and of related work after that is the PhD thesis of Sørensen [76]. The calculus $\lambda{\to}\langle\rangle$ is close to a simply typed variant of the λ-calculus with *non-erasing* reductions of Klop [49]. The difference is mainly in the fact that Klop uses pairs, whereas we use (more general) tuples. Moreover, Klop studies the normalisation of terms $\iota(M)$, where $\iota(M)$ is a term where all erasing redexes (K-redexes) are turned into non-erasing ones (I-redexes). In our formalism, the reduction rules $\bar\beta$ themselves are non-erasing. But most importantly, the use of the new reductions is different: Klop uses the non-erasing reductions to prove that WN for $\bar\beta$ implies SN for β. We are interested in proving SN for $\bar\beta$ itself. We could have used Klop's ideas to prove SN for $\bar\beta$ via WN for $\bar\beta$, but we will not do that and prove SN for $\bar\beta$ directly, because it is shorter.

The last three reduction rules for $\lambda{\to}\langle\rangle$ make sure that a $\langle -\rangle$ pair can move up through the term structure and disappear – as long as the term doesn't contain abstractions. Alternatively phrased, applications can be moved under a $\langle -\rangle$. We make this explicit via the following definition of $\lambda{\to}\langle\rangle$-*context* and lemma.

Definition 4.2.10 *A* $\lambda{\to}\langle\rangle$-context *is given by the following abstract syntax* $K[-]$.

$$K[-] := [-] \mid \mathsf{T}_{\langle\rangle} K[-] \mid K[-]\mathsf{T}_{\langle\rangle}.$$

So a $\lambda{\to}\langle\rangle$-*context* is a $\lambda{\to}\langle\rangle$-term consisting only of applications (no abstractions) with one open place.

Definition 4.2.11 *For every $\lambda{\to}\langle\rangle$ context K and every $\lambda{\to}\langle\rangle$ term T, we define* filling the hole of K with T, *as follows:*

$$\begin{array}{lcllcl} [-][T] & := & T & (S\ K)[T] & := & (S\ (K[T])) \\ (K\ S)[T] & := & ((K[T])\ S) & & & \end{array}$$

where S is a term of $\lambda{\to}\langle\rangle$.

Lemma 4.2.12 *For all $\lambda{\to}\langle\rangle$ contexts K and $\lambda{\to}\langle\rangle$-terms $M, N_1, \ldots, N_k$*

$$K[\langle M, N_1, \ldots, N_k\rangle] \twoheadrightarrow_{\bar{\beta}} \langle K[M], N_1, \ldots, N_k\rangle.$$

Proof By induction to the structure of K.

We now give an interpretation $(\!|-|\!)$ of deduction graphs as $\lambda{\to}\langle\rangle$-terms and we show that a cut-elimination step for graph deductions is mimicked by a (non-empty) $\bar{\beta}$-reduction. Then we show that $\bar{\beta}$-reduction is strongly normalising and we conclude that cut-elimination for deduction graphs is strongly normalising.

Definition 4.2.13 *Given a deduction graph G and a node n in G, we define the λ-term $(\!|G, n|\!)$ as follows (by induction on the number of nodes of G).*

- A *If (n, A) has no outgoing edges, $(\!|G, n|\!) := x^A_n$,*
- E *If $(n, B){\to\!\!\triangleright}(m, A{\to}B)$, and $(n, B){\to\!\!\triangleright}(p, A)$, define $(\!|G, n|\!) := (\!|G, m|\!)\ (\!|G, p|\!)$.*
- R *If $(n, A) \longrightarrow\!\!\triangleright (m, A)$, define $(\!|G, n|\!) := (\!|G, m|\!)$*
- I *If $(n, A{\to}B)$ is a box-node with $(n, A{\to}B) \longrightarrow\!\!\triangleright (j, B)$, the free nodes of the box are $n_1, \ldots, n_k$ and the nodes without incoming edges inside the box are $m_1, \ldots, m_t$, then*

$$(\!|G, n|\!) := \lambda x{:}A.\langle (\!|G, j|\!), (\!|G, m_1|\!), \ldots, (\!|G, m_t|\!)\rangle[x_{n_1} := x, \ldots, x_{n_k} := x].$$

The interpretation of the deduction graph G, $(\!|G|\!)$, is defined as $\langle (\!|G, r_1|\!), \ldots, (\!|G, r_l|\!)\rangle$, where $r_1, \ldots, r_l$ are the top-level nodes without incoming edges in the deduction graph G.

Lemma 4.2.14 (Cut-elimination is $\bar{\beta}$-reduction in $\lambda{\to}\langle\rangle$) *If $G \to_{\mathrm{cut}} G'$, then $(\!|G|\!) \twoheadrightarrow^{+}_{\bar{\beta}} (\!|G'|\!)$.*

Proof By induction on the structure of G. Note that, if G' is obtained from G via a repeat-elimination step, an unsharing step or an incorporation step, then $(\!|G|\!) \equiv_p (\!|G'|\!)$. If G' is obtained from G by contracting a safe cut, say with (p, B) being the conclusion of the $\to$-E, then $(\!|G, p|\!)$ is a subterm of $(\!|G|\!)$, possibly

several times. We look at the subterm $(\!|G,p|\!)$ and we observe the following (keep Figure 4.2 in mind)

$$\begin{array}{lcl}
(\!|G,p|\!) & \equiv & (\!|G,n|\!)(\!|G,m|\!) \\
& \equiv & (\lambda x{:}A.\langle (\!|G,j|\!), (\!|G,m_1|\!), \ldots, (\!|G,m_t|\!)\rangle[x_{n_1} := x, \ldots, x_{n_k} := x]) \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\!|G,m|\!) \\
& \longrightarrow_{\bar\beta} & \langle (\!|G,j|\!), (\!|G,m_1|\!), \ldots, (\!|G,m_t|\!)\rangle[x_{n_1} := (\!|G,m|\!), \ldots, x_{n_k} := (\!|G,m|\!)] \\
& \equiv & \langle (\!|G',p|\!), (\!|G,m_1|\!)^*, \ldots, (\!|G,m_t|\!)^*\rangle
\end{array}$$

where the superscript * denotes the substitution $[x_{n_1} := (\!|G,m|\!), \ldots, x_{n_k} := (\!|G,m|\!)]$. This is the case where $x \in \mathrm{FV}(\langle (\!|G,j|\!), (\!|G,m_1|\!), \ldots, (\!|G,m_t|\!)\rangle[x_{n_1} := x, \ldots, x_{n_k} := x])$, that is where the discharge is not empty.

For the case where

$$x \notin \mathrm{FV}(\langle (\!|G,j|\!), (\!|G,m_1|\!), \ldots, (\!|G,m_t|\!)\rangle[x_{n_1} := x, \ldots, x_{n_k} := x])$$

(an empty discharge), we find

$$\begin{array}{lcl}
(\!|G,p|\!) & \equiv & (\!|G,n|\!)(\!|G,m|\!) \\
& \equiv & (\lambda x{:}A.\langle (\!|G,j|\!), (\!|G,m_1|\!), \ldots, (\!|G,m_t|\!)\rangle[x_{n_1} := x, \ldots, x_{n_k} := x]) \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\!|G,m|\!) \\
& \longrightarrow_{\bar\beta} & \langle\langle (\!|G,j|\!), (\!|G,m_1|\!), \ldots, (\!|G,m_t|\!)\rangle, (\!|G,m|\!)\rangle \\
& \longrightarrow_{\bar\beta} & \langle (\!|G,j|\!), (\!|G,m_1|\!), \ldots, (\!|G,m_t|\!), (\!|G,m|\!)\rangle \\
& \equiv & \langle (\!|G',p|\!), (\!|G,m_1|\!), \ldots, (\!|G,m_t|\!), (\!|G,m|\!)\rangle
\end{array}$$

If we look at $(\!|G,p|\!)$ as a subterm of $(\!|G|\!)$, we first note that it occurs as a subterm of a $\langle \ldots, \ldots \rangle$ expression as follows: $(\!|G|\!) = \langle \ldots, K[(\!|G,p|\!)], \ldots \rangle$.

In a cut-elimination, the box is being removed and the nodes $m_1, \ldots, m_t$ are moved one level up (in terms of depth), thus we find, for the case where the discharge is not empty,

$$\begin{array}{lcl}
(\!|G|\!) & \equiv & \langle \ldots, K[(\!|G,p|\!)], \ldots \rangle \\
& \longrightarrow\!\!\!\!\!\rightarrow^{+}_{\bar\beta} & \langle \ldots, K[\langle (\!|G',p|\!), (\!|G,m_1|\!)^*, \ldots, (\!|G,m_t|\!)^*\rangle], \ldots \rangle \\
& \longrightarrow\!\!\!\!\!\rightarrow_{\bar\beta} & \langle \ldots, \langle K[(\!|G',p|\!)], (\!|G,m_1|\!)^*, \ldots, (\!|G,m_t|\!)^*\rangle, \ldots \rangle \\
& \longrightarrow_{\bar\beta} & \langle \ldots, K[(\!|G',p|\!)], (\!|G,m_1|\!)^*, \ldots, (\!|G,m_t|\!)^*, \ldots \rangle \\
& \equiv_p & (\!|G'|\!)
\end{array}$$

For the case where the discharge is empty we find in a similar way,

$$\begin{array}{lcl}
(\!|G|\!) & \longrightarrow\!\!\!\!\!\rightarrow^{+}_{\bar\beta} & \langle \ldots, K[(\!|G',p|\!)], (\!|G,m_1|\!), \ldots, (\!|G,m_t|\!), (\!|G,m|\!), \ldots \rangle \\
& \equiv_p & (\!|G'|\!)
\end{array}$$

We now prove strong normalisation of $\longrightarrow_{\bar\beta}$. The proof uses an adaptation of the standard method of saturated sets. We interpret types as sets of

(strongly normalising, not necessarily well-typed) terms and we show that this interpretation is sound. All types will be interpreted as a so called *saturated set*.

Definition 4.2.15 *A set X of $\lambda{\to}\langle\rangle$-terms is called* saturated *if*

- $X \subset \mathrm{SN}_{\bar{\beta}}$,
- $\langle x\vec{P}, N_1, \ldots, N_k\rangle \in X$ *(if $k = 0$, this is $x\vec{P}$) for all variables x and all $P_1, \ldots, P_l$, $N_1, \ldots, N_k \in \mathrm{SN}_{\bar{\beta}}$, where $\vec{P} = P_1 \ldots P_l$,*
- *If $M[x := N]\vec{P} \in X$, then $(\lambda x{:}A.M)N\vec{P} \in X$, if $N \in \mathrm{SN}_{\bar{\beta}}$,*
- *If $M \in X$, then $\langle M, P_1, \ldots, P_k\rangle \in X$, if $P_1, \ldots, P_k \in \mathrm{SN}_{\bar{\beta}}$,*
- *If $\langle MN, P_1, \ldots, P_k\rangle\vec{R} \in X$, then $\langle M, P_1, \ldots, P_k\rangle N\vec{R} \in X$.*

We now give an (sound) interpretation of types as saturated sets.

Definition 4.2.16 *The interpretation of types as saturated sets $\mathcal{V}(-)$ is defined as follows.*

- $\mathcal{V}(\alpha) := \mathrm{SN}_{\bar{\beta}}$
- $\mathcal{V}(A{\to}B) := \{M \mid \forall N \in \mathcal{V}(A)(MN \in \mathcal{V}(B))\}$

Lemma 4.2.17 *For all types A, $\mathcal{V}(A)$ is a saturated set.*

Proof This is proved by induction on A. If A is a variable α, $\mathcal{V}(\alpha) = \mathrm{SN}_{\bar{\beta}}$, so we have to check that $\mathrm{SN}_{\bar{\beta}}$ is a saturated set. This is easily checked by verifying that the closure conditions all hold for $\mathrm{SN}_{\bar{\beta}}$. For arrow types we proceed by induction on the structure of types. So suppose that $\mathcal{V}(A)$ and $\mathcal{V}(B)$ are saturated and we have to verify the closure conditions for saturated sets for $\mathcal{V}(A{\to}B)$.

- $\mathcal{V}(A{\to}B)$ is obviously a subset of $\mathrm{SN}_{\bar{\beta}}$ (given that $\mathcal{V}(B) \subset \mathrm{SN}_{\bar{\beta}}$ and that $\mathcal{V}(A) \neq \emptyset$).
- $\langle x\vec{P}, N_1, \ldots, N_k\rangle \in \mathcal{V}(A{\to}B)$ for all variables x and all $P_1, \ldots, P_l$, $N_1, \ldots, N_k \in \mathrm{SN}_{\bar{\beta}}$, where $\vec{P} = P_1 \ldots P_l$, because for all $Q \in \mathcal{V}(A)$ we have $\langle x\vec{P}Q, N_1, \ldots, N_k\rangle \in \mathcal{V}(B)$ and thus $\langle x\vec{P}, N_1, \ldots, N_k\rangle Q \in \mathcal{V}(B)$. So $\langle x\vec{P}, N_1, \ldots, N_k\rangle \in \mathcal{V}(A{\to}B)$.
- Suppose $M[x := N]\vec{P} \in \mathcal{V}(A{\to}B)$. Then for all $Q \in \mathcal{V}(A)$, $M[x := N]\vec{P}Q \in \mathcal{V}(B)$, thus $(\lambda x{:}A.M)N\vec{P}Q \in \mathcal{V}(B)$. So, $(\lambda x{:}A.M)N\vec{P} \in \mathcal{V}(A{\to}B)$.
- Suppose $M \in \mathcal{V}(A{\to}B)$. Then for all $Q \in \mathcal{V}(A)$, $MQ \in \mathcal{V}(B)$, so for all $Q \in \mathcal{V}(A)$ and for all $N_1, \ldots, N_k \in \mathrm{SN}_{\bar{\beta}}$, $\langle MQ, N_1, \ldots, N_k\rangle \in \mathcal{V}(A)$, so for all $Q \in \mathcal{V}(A)$ and for all $N_1, \ldots, N_k \in \mathrm{SN}_{\bar{\beta}}$, $\langle M, N_1, \ldots, N_k\rangle Q \in \mathcal{V}(A)$. We conclude that for all $N_1, \ldots, N_k \in \mathrm{SN}_{\bar{\beta}}$, $\langle M, N_1, \ldots, N_k\rangle \in \mathcal{V}(A{\to}B)$.

- Suppose $\langle MN, P_1, \ldots, P_k\rangle\vec{R} \in \mathcal{V}(A{\to}B)$, then for all $Q \in \mathcal{V}(A)$, $\langle MN, P_1, \ldots, P_k\rangle\vec{R}Q \in \mathcal{V}(B)$. So for all $Q \in \mathcal{V}(A)$, $\langle M, P_1, \ldots, P_k\rangle N\vec{R}Q \in \mathcal{V}(B)$ and we can conclude that $\langle M, P_1, \ldots, P_k\rangle N\vec{R} \in \mathcal{V}(A{\to}B)$.

Lemma 4.2.18 (Soundness of $\mathcal{V}(-)$) *If $x_1 : A_1, \ldots, x_n : A_n \vdash M : B$ in $\lambda{\to}\langle\rangle$ and $N_1 \in \mathcal{V}(A_1), \ldots N_n \in \mathcal{V}(A_n)$, then $M[x_1 := N_1, \ldots, x_n := N_n] \in \mathcal{V}(B)$.*

Proof By induction on the derivation. All cases are straightforward; as illustration we treat two. For simplicity we denote $M[x_1 := N_1, \ldots, x_n := N_n]$ as M^*.

abs So, $\Gamma \vdash \lambda x{:}A.M : A{\to}B$ was derived from $\Gamma, x{:}A \vdash M : B$. As IH we have $\forall Q \in \mathcal{V}(A)(M^*[x := Q] \in \mathcal{V}(B))$. One of the closure conditions of saturated sets then says that $\forall Q \in \mathcal{V}(A)((\lambda x{:}A.M^*)Q \in \mathcal{V}(B))$ and therefore $\lambda x{:}A.M^* \in \mathcal{V}(A{\to}B)$.

$\langle -, \ldots\rangle$ So, $\Gamma \vdash \langle M, P_1, \ldots, P_k\rangle : A$ was derived from $\Gamma \vdash M : A$ and $\Gamma \vdash P_1 : B_1$, $\ldots$, $\Gamma \vdash P_k : B_k$. As IH we have $M^* \in \mathcal{V}(A)$ and $P_i^* \in \mathcal{V}(B_i)$ for $1 \leq i \leq k$ (and thus $P_i^* \in \mathrm{SN}_{\bar{\beta}}$ for $1 \leq i \leq k$). One of the closure conditions of saturated sets then says that $\langle M, P_1, \ldots, P_k\rangle^* \in \mathcal{V}(A)$.

Corollary 4.2.19 *$\bar{\beta}$-reduction is strongly normalising for $\lambda{\to}\langle\rangle$ and thus cut-elimination is terminating for deduction graphs.*

Proof Take $N_i := x_i$ in the Lemma and we find that $M \in \mathcal{V}(B) \subset \mathrm{SN}_{\bar{\beta}}$, so M is strongly normalising. Henceforth cut-elimination is terminating for deduction graphs due to Lemma 4.2.14.

4.3 Context Calculus

4.3.1 λ + let-calculus

As stated in Lemma 4.2.4, the mapping $[\![-]\!]$ from deduction graphs to λ-terms does not reflect the structure of the deduction graph very well: it may ignore parts and does not reflect sharing in the term. The mapping $\langle\![-]\!\rangle$ does a bit better, as all parts of the deduction graph are reflected in the term, but it also introduces duplications, because it does not take sharing into account. We would like a term:

- to show the structure of the whole graph G, instead of only the part that is relevant for some node n. This would be useful in the case of e.g. an $\to$-E rule, where we want to combine two parts of an existing graph.
- to show which parts of G are shared.
- to show which parts of G are repeated.

- to show how boxes are positioned with respect to each other.

To get a more faithful term-representation of deduction graphs, we define a mapping to a context calculus with let-expressions. As we want to represent the whole graph, we cannot take one node as a "focus" and define the translation from there (as we did for $[\![-]\!]$ in Definition 4.2.1). A context is a term with a "hole", an open place where we can fill in a term. If we fill in x_n (where n is a top-level node of G), we get the interpretation of G as a term *in node* n.

The translation from deduction graphs to contexts is a bit involved, and so are the reduction rules on contexts that correspond to the reduction rules on deduction graphs. This is to be expected, because we want to represent a graph in a faithful way as an expression. On the other hand, this translation to contexts also clarifies a couple of things. First of all, it clarifies the translation $[\![-]\!]$ to simply typed terms, as it can be factored through the translation to contexts that we will define now. (This is made precise in Section 4.5.) Furthermore, the translation to contexts clarifies the connection with let calculi, e.g. see [59]. Our contexts have 'let declarations' of the form $\mathsf{let}\, x = P \,\mathsf{in}\, Q$, which are used to represent the sharing. We could also have chosen to formulate it with *explicit substitutions*, i.e. $Q\langle x = P\rangle$: P is to be substituted for x in Q. The reduction rules on our contexts show how the reduction rules on deduction graphs can be understood as a let-calculus or an explicit substitution calculus.

Before giving the interpretation, we make the calculus $\lambda{\to}{+}\mathsf{let}$ precise. We first define the set of *expressions* and the operation of *filling holes*. Then we define the subset of *terms* and *contexts*.

Definition 4.3.1 *The syntactic domain of expressions is:*

$$\textit{Expressions} \quad E \;:=\; x \mid (xy) \mid [-] \mid \mathsf{let}\, x = E \,\mathsf{in}\, E \mid \lambda x{:}A.E$$

Definition 4.3.2 *For every two expressions E_1 and E_2 we define* filling the holes of E_1 with E_2, $E_1[E_2]$, *as follows:*

$$\begin{array}{lcllcl} x[E] & := & x & (\mathsf{let}\, x = F \,\mathsf{in}\, G)[E] & := & \mathsf{let}\, x = (F[E]) \,\mathsf{in}\, (G[E]) \\ xy[E] & := & xy & (\lambda x.F)[E] & := & \lambda x.(F[E]) \\ [-][E] & := & E & & & \end{array}$$

where E, F and G are expressions.

Definition 4.3.3 *The calculus of contexts $\lambda{\to}{+}\mathsf{let}$ is defined as follows. Types are the simple types,* Typ. *The syntactic domains are:*

$$\textit{Terms} \quad T \;:=\; x \mid (xy) \mid \lambda x{:}A.C[y]$$

$$\textit{Contexts} \quad C \;:=\; [-] \mid \mathsf{let}\, x = T \,\mathsf{in}\, C$$

A context C has exactly one 'hole', $[-]$. We sometimes write $C[-]$ to emphasize this. In the syntax for Terms, $C[y]$ denotes the filling of the unique hole in C with y.

Typing Rules for Terms and Contexts. As usual, Γ *is the environment consisting of term variable declarations:* $x_1 : A_1, \ldots, x_n : A_n$*, where all the* x_i *are distinct. In addition we have a* let-environment Δ*, which denotes the sequence of typed let-bound variables that surrounds a context-hole. There are two judgments* $\Gamma \vdash M : A$ *(*the term M is of type A*) and* $\Gamma; \Delta \vdash C$ *(*C is a well-formed context*).*

$$\Gamma; <> \vdash [-] \qquad \frac{\Gamma \vdash T : A \;\; \Gamma, x{:}A; \Delta \vdash C}{\Gamma; x{:}A, \Delta \vdash \mathsf{let}\, x = T \,\mathsf{in}\, C}\,\mathsf{let}$$

$$\Gamma \vdash x : A \quad \textit{if } (x{:}A) \in \Gamma \qquad \frac{\Gamma, x{:}A; \Delta \vdash C}{\Gamma \vdash \lambda x{:}A.C[y] : A{\to}B}\,\lambda \quad \textit{if } (y{:}B) \in \Delta$$

$$\frac{\Gamma \vdash x : A{\to}B \;\; \Gamma \vdash y : A}{\Gamma \vdash xy : B}\,app$$

We now list some properties that give an intuition for the $\lambda{\to}+\mathsf{let}$-calculus. The proofs are by a straightforward induction on the derivation.

Lemma 4.3.4

1. *Shape of well-formed contexts. If*

$$\Gamma; x_1{:}A_1, \ldots, x_n{:}A_n \vdash C[-]$$

then

$$C[-] \equiv \mathsf{let}\, x_1 = t_1 \,\mathsf{in}\, \ldots \mathsf{let}\, x_n = t_n \,\mathsf{in}\, [-]$$

with $\Gamma \vdash t_i : A_i$ *(for* $1 \leq i \leq n$*).*

2. *Substitution of variables. If*

$$\Gamma, x : A, \Gamma'; \Delta \vdash C[-] \textit{ and } y{:}A \in \Gamma, \Gamma'$$

then

$$\Gamma, \Gamma'; \Delta \vdash C[-][x := y]$$

and similarly for terms: If

$$\Gamma, x : A, \Gamma' \vdash M \textit{ and } y{:}A \in \Gamma, \Gamma'$$

then

$$\Gamma, \Gamma' \vdash M[x := y]$$

3. *Substitution of contexts: if*

$$\Gamma; \Delta, \Theta \vdash C[-] \textit{ and } \Gamma, \Theta; \Lambda \vdash D[-]$$

then

$$\Gamma; \Delta, \Theta, \Lambda \vdash C[D[-]]$$

In the translation $\langle\!\langle G \rangle\!\rangle$, separate parts of the process of cut-elimination can be recognized. We now define a reduction relation on $\lambda\!\rightarrow$ + let-contexts that captures cut-elimination on deduction graphs.

Definition 4.3.5 *We define the* free variables *of expressions E, $FV(E)$, as follows:*

$$\begin{array}{lcllcl} FV(x) & := & \{x\} & FV(\mathsf{let}\, x = E_1 \,\mathsf{in}\, E_2) & := & FV(E_1) \cup (FV(E_2) \setminus \{x\}) \\ FV(xy) & := & \{x, y\} & FV(\lambda x.E) & := & FV(E) \setminus \{x\} \\ FV([-]) & := & \emptyset & & & \end{array}$$

Definition 4.3.6 *We define the following (conditional) rewrite rules for expressions.*

$$\begin{array}{rcll} \mathsf{let}\, x = \lambda z{:}A.D[y] \,\mathsf{in}\, \mathsf{let}\, p = (xq) \,\mathsf{in}\, E & \longrightarrow_B & D[z := q][\mathsf{let}\, p = y \,\mathsf{in}\, E] & \\ & & \text{if } D \text{ a context, } x \notin FV(E) & \\ \mathsf{let}\, x = P \,\mathsf{in}\, \mathsf{let}\, y = \lambda z.Q \,\mathsf{in}\, E & \longrightarrow_{CM} & \mathsf{let}\, y = \lambda z.(\mathsf{let}\, x = P \,\mathsf{in}\, Q) \,\mathsf{in}\, E & \\ & & \text{if } x \notin FV(E) & \\ \mathsf{let}\, x = P \,\mathsf{in}\, E[y := x] & \longrightarrow_{CP} & \mathsf{let}\, y = P \,\mathsf{in}\, \mathsf{let}\, x = P \,\mathsf{in}\, E & \\ & & \text{if } x, y \in FV(E) & \\ \mathsf{let}\, y = x \,\mathsf{in}\, E & \longrightarrow_{L} & E[y := x] \quad \text{if } E \neq C[y] & \\ & & \text{for any context } C & \\ \mathsf{let}\, x = P \,\mathsf{in}\, \mathsf{let}\, y = Q \,\mathsf{in}\, E & \simeq & \mathsf{let}\, y = Q \,\mathsf{in}\, \mathsf{let}\, x = P \,\mathsf{in}\, E & \\ & & \text{if } x \notin FV(Q), y \notin FV(P) & \end{array}$$

We extend the rewrite rules to a reduction relation on expressions by taking the contextual closure, thus, if $E \longrightarrow_B E'$, then

$$\begin{array}{rcl} \lambda x{:}A.E & \longrightarrow_B & \lambda x{:}A.E' \\ \mathsf{let}\, x = T \,\mathsf{in}\, E & \longrightarrow_B & \mathsf{let}\, x = T \,\mathsf{in}\, E', \\ \mathsf{let}\, x = E \,\mathsf{in}\, T & \longrightarrow_B & \mathsf{let}\, x = E' \,\mathsf{in}\, T. \end{array}$$

Similarly for the other rewrite rules and for $\simeq$.

We write $\longrightarrow_{\mathrm{let}}$ for the union of these reduction rules.

These rewrite rules are meant to describe the transformations on deduction graphs. We will see later that the $\longrightarrow_B$-rule corresponds to safe cut-elimination. The context D is then associated to the nodes in the box $\mathcal{B}$, which move up one level because $\mathcal{B}$ disappears. The rule $\longrightarrow_{CM}$, to give another example, corresponds to incorporation.

Remark 4.3.7 *Note that the contextual closure of the rewrite rules that we use in the previous definition takes side conditions into account that concern the free variables.*

As a consequence, it may occur that $C[-] \longrightarrow_B D[-]$ but not *$\lambda u{:}B.C[x] \longrightarrow_B \lambda u{:}B.D[x]$, because $x \notin FV(C[-])$, allowing $C[-]$ to reduce, but $x \in FV(C[x])$, disallowing $\lambda u{:}B.C[x]$ to reduce. This situation occurs for example in the term*

$$t := \lambda u{:}B.(\mathsf{let}\, x = (\lambda z{:}A.z) \,\mathsf{in}\, \mathsf{let}\, p = (xq) \,\mathsf{in}\, x).$$

The context $C[-] = \mathsf{let}\, x = (\lambda z{:}A.z)\, \mathsf{in}\, \mathsf{let}\, p = (xq)\, \mathsf{in}\, [-]$ *can be reduced, but the term* $t := \lambda u{:}B.C[x]$ *cannot (and rightly so).*

Conversely, it may occur that $\lambda u{:}B.C[x] \twoheadrightarrow_{CP} \lambda u{:}B.D[x]$*, but* not $C[-] \twoheadrightarrow_{CP} D[-]$*, because* $x \notin FV(C[-])$*, disallowing* $C[-]$ *to reduce, but* $x, y \in FV(C[x])$*, allowing* $\lambda u{:}B.C[x]$ *to reduce. This situation occurs for example in the term*

$$t := \lambda u{:}B.\mathsf{let}\, x = x_1x_2\, \mathsf{in}\, \mathsf{let}\, x_3 = x_4x\, \mathsf{in}\, x,$$

which CP-reduces to $\lambda u{:}B.\mathsf{let}\, y = x_1x_2\, \mathsf{in}\, \mathsf{let}\, x = x_1x_2\, \mathsf{in}\, \mathsf{let}\, x_3 = x_4y\, \mathsf{in}\, x$*, whereas the context* $\mathsf{let}\, x = x_1x_2\, \mathsf{in}\, \mathsf{let}\, x_3 = x_4x\, \mathsf{in}\, [-]$ *does not reduce (and rightly so).*

Remark 4.3.8 *The rule* $\twoheadrightarrow_{CP}$ *makes reduction of expressions non-deterministic. Consider for example the following expression:*

$$\mathsf{let}\, x = z\, \mathsf{in}\, \mathsf{let}\, u = (xp)\, \mathsf{in}\, \mathsf{let}\, v = (xw)\, \mathsf{in}\, x$$

This reduces to

$$\mathsf{let}\, y = z\, \mathsf{in}\, \mathsf{let}\, x = z\, \mathsf{in}\, \mathsf{let}\, u = (yp)\, \mathsf{in}\, \mathsf{let}\, v = (xw)\, \mathsf{in}\, x$$

as well as to

$$\mathsf{let}\, y = z\, \mathsf{in}\, \mathsf{let}\, x = z\, \mathsf{in}\, \mathsf{let}\, u = (yp)\, \mathsf{in}\, \mathsf{let}\, v = (xw)\, \mathsf{in}\, y$$

although we have contracted the same redex. In both cases we match x *with* x*,* z *with* P*, and* $\mathsf{let}\, u = (xp)\, \mathsf{in}\, \mathsf{let}\, v = (xw)\, \mathsf{in}\, x$ *with* $E[y := x]$*.*

4.3.2 Subject Reduction

We now want to show that the well-formed terms and contexts are closed under reduction. Under reduction, a context may get transformed quite a bit, with as consequence that the Δ in which the context is typable has to be transformed. We now 'fix' a context in which a context is well-formed.

Definition 4.3.9 *Let* $C = \mathsf{let}\, x_1 = t_1\, \mathsf{in}\, \ldots \mathsf{let}\, x_n = t_n\, \mathsf{in}\, [-]$ *be a well-formed context in* $\Gamma; \Delta$ *and let* T *be a well-formed term in* Γ*. The* let-context *of* C *is* $x_1{:}A_1, \ldots, x_n{:}A_n$*, where* A_i *is the type of* x_i *in* Δ*. We denote the let-context of* C *by* Δ^C*.*

Lemma 4.3.10 (Subject Reduction)
If $\Gamma; \Delta \vdash C$ *and* $C \twoheadrightarrow_{\mathrm{let}} D$*, then* $\Gamma; \Delta^D \vdash D$*. If* $\Gamma \vdash M : A$ *and* $M \twoheadrightarrow_{\mathrm{let}} N$*, then* $\Gamma \vdash N : A$*.*

To prove the subject reduction property, it is convenient to change the typing rules a little, giving also a type to the subexpression $C[y]$ (which is not well-typed in $\lambda{\to} + \mathsf{let}$). Thus, we change the λ-rule into the following two rules: the *fill-rule* and the *abs-rule*.

$$\frac{\Gamma;\Delta \vdash C}{\Gamma \vdash C[y] : B}\ \text{fill-rule}\quad \text{if } (y{:}B) \in \Delta \qquad \frac{\Gamma, x{:}A \vdash C[y] : B}{\Gamma \vdash \lambda x{:}A.C[y] : A{\to}B}\ \text{abs-rule}$$

If we call the new systems $\lambda{\to} + \mathsf{let}^+$ and denote derivability in this system by $\vdash^+$, we immediately see that

$$\begin{array}{rcl} \Gamma;\Delta \vdash^+ C & \Leftrightarrow & \Gamma;\Delta \vdash C \\ \Gamma \vdash^+ t : A & \Leftarrow & \Gamma \vdash t : A \\ \Gamma \vdash^+ t : A \wedge t \neq C[y] & \Rightarrow & \Gamma \vdash t : A \end{array}$$

We will now prove SR for $\lambda{\to} + \mathsf{let}^+$ and we conclude SR for $\lambda{\to} + \mathsf{let}$ by observing that, if $\Gamma \vdash t : A$ and $t \twoheadrightarrow_{\mathrm{let}} q$, then $q \neq C[y]$.

Apart from the meta-theoretic properties of $\lambda{\to}{+}\mathsf{let}$ already listed in Section 4.3.1, the proof of subject reduction also uses the following ones:

Lemma 4.3.11

1. *Weakening: if* $\Gamma;\Delta \vdash^+ C$ *and* $\Gamma \subseteq \Gamma'$*, then* $\Gamma';\Delta \vdash^+ C$ *and similarly for terms.*

2. *Strengthening: if* $\Gamma;\Delta \vdash^+ C$*, then* $\Gamma\lceil FV(C);\Delta \vdash^+ C$ *and similarly for terms.*

3. *Generation: a well-formed context or a well-typed term can only have been created in one way (see the derivation rules).*

Proof By induction on the structure of the expression.

Proof [of Lemma 4.3.10] By induction on the structure of the expression, distinguishing cases according to the reduction rule. For the proof of subject reduction, there are four base cases, where $\twoheadrightarrow_B$, $\twoheadrightarrow_{CM}$, $\twoheadrightarrow_{CP}$ or $\twoheadrightarrow_L$ is applied to a context on the "top level". And then there are four contextual closure cases.

Base We treat the base cases where $\twoheadrightarrow_B$ or $\twoheadrightarrow_{CM}$ is applied to a context on the "top level". Below we use $\Delta_{x,y}$ to denote Δ with the declarations of x and y removed.

$\twoheadrightarrow_B$ $\mathsf{let}\, x = \lambda z{:}A.D[y]\, \mathsf{in}\, \mathsf{let}\, p = xq\, \mathsf{in}\, C \twoheadrightarrow_B D[z := q][\mathsf{let}\, p = y\, \mathsf{in}\, C]$, where $x \notin \mathrm{FV}(C)$, we find that

1. $\Gamma, z{:}A;\Delta^D \vdash^+ D$ and $q{:}A \in \Gamma$ and $y{:}E \in \Delta^D$, therefore $\Gamma;\Delta^D \vdash^+ D[z := q]$
2. $\Gamma, x{:}A{\to}E, p{:}E;\Delta_{x,p} \vdash^+ C$ and so $\Gamma, p{:}E;\Delta_{x,p} \vdash^+ C$ because $x \notin \mathrm{FV}(C)$.

From the second we conclude that $\Gamma, y{:}E, p{:}E; \Delta_{x,p} \vdash^+ C$ and thus $\Gamma, y{:}E; p{:}E, \Delta_{x,p} \vdash^+ \mathsf{let}\, p = y \,\mathsf{in}\, C$. From this, using context substitution and the first, we derive $\Gamma; \Delta^D, p{:}E, \Delta_{x,p} \vdash^+ D[z := q][\mathsf{let}\, p = y \,\mathsf{in}\, C]$.

$\twoheadrightarrow_{CM}$ $\mathsf{let}\, x = P \,\mathsf{in}\, \mathsf{let}\, y = \lambda z.D[q] \,\mathsf{in}\, C \twoheadrightarrow_{CM} \mathsf{let}\, y = \lambda z.(\mathsf{let}\, x = P \,\mathsf{in}\, D[q]) \,\mathsf{in}\, C$, where $x \notin \mathrm{FV}(C)$ we find that

1. $\Gamma, x{:}A, y{:}B{\to}E; \Delta_{x,y} \vdash^+ C$ and therefore $\Gamma, y{:}B{\to}E; \Delta_{x,y} \vdash^+ C$ because $x \notin \mathrm{FV}(C)$.
2. $\Gamma, x{:}A, z{:}B; \Delta^D \vdash^+ D$ with $q : E \in \Delta^D$ and therefore $\Gamma, z{:}B, x{:}A; \Delta^D \vdash^+ D$.
3. $\Gamma \vdash^+ P : A$ and therefore $\Gamma, z{:}B \vdash^+ P : A$.

The second and third yield $\Gamma \vdash^+ \lambda z{:}B.\mathsf{let}\, x = P \,\mathsf{in}\, D[q] : B{\to}E$. So, using the first we derive $\Gamma; \Delta_x \vdash^+ \mathsf{let}\, y = \lambda z{:}B.(\mathsf{let}\, x = P \,\mathsf{in}\, D[q]) \,\mathsf{in}\, C$.

Closure

1. $\lambda x{:}A.P \twoheadrightarrow \lambda x{:}A.P'$, because $P \twoheadrightarrow P'$; this case is easy.
2. $\mathsf{let}\, x = P \,\mathsf{in}\, C \twoheadrightarrow \mathsf{let}\, x = P' \,\mathsf{in}\, C$, because $P \twoheadrightarrow P'$; this case is also easy.
3. $\mathsf{let}\, x = P \,\mathsf{in}\, C \twoheadrightarrow \mathsf{let}\, x = P \,\mathsf{in}\, C'$, because $C \twoheadrightarrow C'$; this case is also easy.
4. $C[v] \twoheadrightarrow D[w]$. This can be caused by a reduction in $C[v]$ on the "top level": this is the interesting case, to be treated below. If $C[v] = \mathsf{let}\, x_1 = P_1 \ldots \mathsf{let}\, x_n = P_n \,\mathsf{in}\, v \twoheadrightarrow D[w]$ because $P_i \twoheadrightarrow P_i'$, we easily conclude by induction.

For the contextual closure we only consider the most interesting case, where $C[v] \twoheadrightarrow E$ on the top level. There are three subcases:

- $C[v] \twoheadrightarrow_B D[v]$ or $C[v] \twoheadrightarrow_{CM} D[v]$. Note that in D, v is still a let-abstracted variable. Now, $C[-] \twoheadrightarrow D[-]$, so by induction hypothesis $\Gamma; \Delta^D \vdash^+ D[-]$, and $v : B$ must be in Δ^D, so $\Gamma \vdash^+ D[v] : B$.
- $C[v] \twoheadrightarrow_{CP} D[v]$. Then $C[v] = \mathsf{let}\, x = P \,\mathsf{in}\, E[y := x]$, and we can write $E[y := x]$ as $F[y := x][v]$. So $\Gamma, x{:}A; \Delta^{F[y:=x]} \vdash^+ F[y := x]$ with $v : B \in \Delta^{F[y:=x]}$ or $v = x$. We conclude that $\Gamma, y{:}A, x{:}A; \Delta^F \vdash^+ F$ and thus $\Gamma \vdash^+ \mathsf{let}\, y = P \,\mathsf{in}\, \mathsf{let}\, x = P \,\mathsf{in}\, E$.
- $C[v] \twoheadrightarrow_L D[v]$. Then $C[v] = \mathsf{let}\, y = x \,\mathsf{in}\, G[v]$, and we know that $\Gamma, y{:}A \vdash^+ G[v] : B$, with $x{:}A \in \Gamma$. By substitution, we conclude $\Gamma \vdash^+ G[v][y := x] : B$ and we are done.

4.4 Translation to Contexts

4.4.1 Definition of the Translation

In this section we define a translation from deduction graphs to the $\lambda{\to} + \mathsf{let}$-calculus. The definition uses a box-topological ordering. We will see in Section 4.4.2 that the choice of box-topological ordering is not very relevant.

Definition 4.4.1 *Given a deduction graph G and a box-topological ordering φ of G we define a context in $\lambda{\to}{+}\mathsf{let}$ $\langle\!\langle G \rangle\!\rangle_\varphi$ as follows, by induction on the number of nodes and boxes of G.*

- *If $|G| = 1$ (G has only one node), then*

$$\langle\!\langle G \rangle\!\rangle_\varphi = [-].$$

- *If $|G| > 1$ and the φ-maximal element, say n, is an* A*-node, then*

$$\langle\!\langle G \rangle\!\rangle_\varphi = \langle\!\langle G \setminus n \rangle\!\rangle_{\varphi_{|G\setminus n}}$$

- *If the φ-maximal element, say n, is an* I*-node with $(n, A{\to}B) \longrightarrow\!\triangleright (j, B)$ with associated box $\mathcal{B}$, then*

$$\langle\!\langle G \rangle\!\rangle_\varphi = \langle\!\langle G \setminus \mathcal{B} \rangle\!\rangle_{\varphi_{|G\setminus\mathcal{B}}}[\mathsf{let}\, x_n = (\lambda x_m{:}A.\langle\!\langle \mathcal{B}^m \rangle\!\rangle_{\varphi_{|\mathcal{B}^m}}[x_j])\, \mathsf{in}\, [-]],$$

  where $\varphi_{|\mathcal{B}^m}$ is $\varphi_{|\mathcal{B}}$ with an extra minimal element m (m is fresh);

- *If the φ-maximal element, say n is an* E*-node with $(n, B) \longrightarrow\!\triangleright (i, A{\to}B)$ and $(n, B) \longrightarrow\!\triangleright (j, A)$, then*

$$\langle\!\langle G \rangle\!\rangle_\varphi := \langle\!\langle G \setminus n \rangle\!\rangle_{\varphi_{|G\setminus n}}[\mathsf{let}\, x_n = (x_i x_j)\, \mathsf{in}\, [-]];$$

- *If the φ-maximal element, say n, is an* R*-node with $n \longrightarrow\!\triangleright l$, then*

$$\langle\!\langle G \rangle\!\rangle_\varphi = \langle\!\langle G \setminus n \rangle\!\rangle_{\varphi_{|G\setminus n}}[\mathsf{let}\, x_n = x_l\, \mathsf{in}\, [-]].$$

This translation 'turns the structure inside out': the variables corresponding to nodes that are first reached from the maximal node, will be deepest in the term.

Example 4.4.2 *We give the associated contexts in $\lambda{\to}{+}\mathsf{let}$ for the three simple deduction graphs of Figure 2.5 for the box-topological orderings $1 < 2$, $1 < 2 < 3$, and $1 < 2 < 4 < 3 < 5$ respectively. Note that in example (III), where 'garbage' occurs inside a box, the $\mathsf{let}\, x_3 = \ldots$ declaration corresponds to the piece of 'garbage'. This declaration is under the λ-abstraction $(\lambda x_p{:}B.\ldots)$.*

The translations of the deduction graphs of Figure 2.5, together with their context:

$$\begin{array}{lll}
I & <>; x_2 : A{\to}A & \mathsf{let}\, x_2 = (\lambda x_m{:}A.\mathsf{let}\, x_1 = x_m\, \mathsf{in}\, x_1)\, \mathsf{in}\, [-] \\
II & x_1 : B; x_3 : A{\to}B & \mathsf{let}\, x_3 = (\lambda x_m{:}A.\mathsf{let}\, x_2 = x_1\, \mathsf{in}\, x_2)\, \mathsf{in}\, [-] \\
III & <>; x_5 : B{\to}B & \mathsf{let}\, x_5 = (\lambda x_p{:}B.\mathsf{let}\, x_1 = x_p\, \mathsf{in}\, \mathsf{let}\, x_4 = x_1\, \mathsf{in} \\
 & & \qquad \mathsf{let}\, x_3 = (\lambda x_m{:}A.\mathsf{let}\, x_2 = x_1\, \mathsf{in}\, x_2)\, \mathsf{in}\, x_4)\, \mathsf{in}\, [-]
\end{array}$$

Lemma 4.4.3 (Soundness of $\langle\!\langle - \rangle\!\rangle$) *If G is a deduction graph and φ is a box-topological ordering on G, then*

$$\Gamma; \Delta \vdash \langle\!\langle G \rangle\!\rangle_{\varphi},$$

where $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ with $(1, A_1), \ldots, (n, A_n)$ the free nodes of G, and $\Delta = y_k : B_k, \ldots, y_{k+m} : B_{k+m}$ with $(k, B_k), \ldots, (k+m, B_{k+m})$ the non-free top-level nodes of G.

Proof By induction on the definition of $\langle\!\langle G \rangle\!\rangle_{\varphi}$. Let Γ be a sequence of variable declarations corresponding to the free nodes of G and let Δ be the sequence of non-free top-level nodes of G (in the order given by φ).

- Suppose G has just one node. We easily check that $\Gamma; <> \vdash [-]$.
- Suppose $|G| > 1$ and the φ-maximal element, say n, is an A-node. By induction we get that $\Gamma \setminus x_n; \Delta \vdash \langle\!\langle G \setminus n \rangle\!\rangle_{\varphi_{|G\setminus n}}$. By Weakening we conclude: $\Gamma; \Delta \vdash \langle\!\langle G \setminus n \rangle\!\rangle_{\varphi_{|G\setminus n}} = \langle\!\langle G \rangle\!\rangle_{\varphi}$.
- Suppose the φ-maximal element, say n, is an I-node with $(n, A{\to}B) \longrightarrow\!\triangleright (j, B)$ with associated box $\mathcal{B}$. Suppose the free nodes of $\mathcal{B}^m$ are not free nodes of G. We will write $\vec{x}_{\mathcal{B}^m}$ for the variables corresponding to the free nodes of $\mathcal{B}^m$ and we will write $\vec{y}_{\mathcal{B}^m}$ for the variables corresponding to the non-free top-level nodes of $\mathcal{B}^m$. Then by induction: $x_m{:}A, \vec{x}_{\mathcal{B}^m}; \vec{y}_{\mathcal{B}^m} \vdash \langle\!\langle \mathcal{B}^m \rangle\!\rangle_{\varphi_{|\mathcal{B}^m}}$. So by the λ-rule: $\vec{x}_{\mathcal{B}^m} \vdash \lambda x_m{:}A.\langle\!\langle \mathcal{B}^m \rangle\!\rangle_{\varphi_{|\mathcal{B}^m}}[x_j]{:}A{\to}B$. Because $x_n{:}A, \vec{x}_{\mathcal{B}^m}; <> \vdash [-]$, the let-rule gives us : $\vec{x}_{\mathcal{B}^m}; x_n{:}A \vdash \mathsf{let}\, x_n = \lambda x_m{:}A.\langle\!\langle \mathcal{B}^m \rangle\!\rangle_{\varphi_{|\mathcal{B}^m}}[x_j]\, \mathsf{in}\, [-]$. By Weakening:
  $\Gamma, \vec{x}_{\mathcal{B}^m}; x_n{:}A \vdash \mathsf{let}\, x_n = \lambda x_m{:}A.\langle\!\langle \mathcal{B}^m \rangle\!\rangle_{\varphi_{|\mathcal{B}^m}}[x_j]\, \mathsf{in}\, [-]$. Furthermore we get by induction that $\Gamma; (\Delta \setminus x_n, \vec{x}_{\mathcal{B}^m}), \vec{x}_{\mathcal{B}^m} \vdash \langle\!\langle G \setminus \mathcal{B} \rangle\!\rangle_{\varphi_{G\setminus\mathcal{B}}}$. Hence by Substitution: $\Gamma; (\Delta \setminus x_n, \vec{x}_{\mathcal{B}^m}), \vec{x}_{\mathcal{B}^m}, x_n \vdash \langle\!\langle G \setminus \mathcal{B} \rangle\!\rangle_{\varphi_{G\setminus\mathcal{B}}}[\mathsf{let}\, x_n = \lambda x_m{:}A.\langle\!\langle \mathcal{B}^m \rangle\!\rangle_{\varphi_{|\mathcal{B}^m}}[x_j]\, \mathsf{in}\, [-]]$. That is: $\Gamma; \Delta \vdash \langle\!\langle G \rangle\!\rangle_{\varphi}$. The case that some of the free nodes of $\mathcal{B}^m$ are also free nodes of G, is similar.
- Suppose the φ-maximal element, say n is an E-node with $(n, B) \longrightarrow\!\triangleright (i, A{\to}B)$ and $(n, B) \longrightarrow\!\triangleright (j, A)$. Suppose i and j are not free nodes. By induction we see that $\Gamma \vdash (\Delta \setminus x_n, x_i, x_j), x_j, x_i \vdash \langle\!\langle G \setminus n \rangle\!\rangle_{\varphi_{|G\setminus n}}$. Furthermore $x_j, x_i; x_n \vdash \mathsf{let}\, x_n = (x_i x_j)\, \mathsf{in}\, [-]$. So, by Weakening: $\Gamma, x_j, x_i; x_n \vdash \mathsf{let}\, x_n = (x_i x_j)\, \mathsf{in}\, [-]$. By Substitution we conclude that
  $\Gamma; (\Delta \setminus x_n, x_i, x_j), x_j, x_i, x_n \vdash \langle\!\langle G \setminus n \rangle\!\rangle_{\varphi_{|G\setminus n}}[\mathsf{let}\, x_n = (x_i x_j)\, \mathsf{in}\, [-]]$. That is: $\Gamma; \Delta \vdash \langle\!\langle G \rangle\!\rangle_{\varphi}$. The case that either i or j or both are a free node of G is similar.
- Suppose the φ-maximal element, say n, is an R-node with $n \longrightarrow\!\triangleright l$. Suppose l is not a free node. By induction we see that $\Gamma; (\Delta \setminus x_n, x_l), x_l \vdash \langle\!\langle G \setminus n \rangle\!\rangle_{\varphi_{G\setminus\varphi}}$. Furthermore $x_l; x_n \vdash \mathsf{let}\, x_n = x_l\, \mathsf{in}\, [-]$. So by Weakening: $\Gamma, x_l; x_n \vdash \mathsf{let}\, x_n = x_l\, \mathsf{in}\, [-]$. By Substitution we conclude: $\Gamma; (\Delta \setminus x_n, x_l), x_l, x_n \vdash \langle\!\langle G \setminus n \rangle\!\rangle_{\varphi_{G\setminus\varphi}}[\mathsf{let}\, x_n = x_l\, \mathsf{in}\, [-]]$. That is: $\Gamma; \Delta \vdash \langle\!\langle G \rangle\!\rangle_{\varphi}$. The case that l is a free node of G is similar.

So the interpretation of G as a $\lambda\!\rightarrow\!+\,\mathsf{let}$ context yields a well-formed context indeed.

Example 4.4.4 *We give the interpretations of the deduction graphs in Figure 2.6 (G) and Figure 3.7 (G′) for the box-topological orderings* $3<1<2<4<5<6<N_H<8$*, where* N_H *denotes the nodes of* H*, and* $3<7<1<2<4<5<6<8<9$ *respectively, including the contexts they are well-formed in. We assume that* $\langle\!\langle H\rangle\!\rangle = C[-]$ *in the context* $\Gamma;\Delta$*.*

$$\begin{array}{rcl}
\langle\!\langle G\rangle\!\rangle & = & \mathsf{let}\, x_6 = (\lambda x{:}A.\mathsf{let}\, x_1 = x, x_2 = x, x_4 = (x_3x_2), x_5 = (x_4x_1)\,\mathsf{in}\, x_5)\,\mathsf{in} \\
& & C[\mathsf{let}\, x_8 = (x_6x_7)\,\mathsf{in}\,[-]] \\
& & \textit{in context}\ \Gamma \cup x_3{:}A{\rightarrow}A{\rightarrow}B;\Delta \cup x_7{:}(A{\rightarrow}B){\rightarrow}A, x_6{:}A{\rightarrow}B, x_8{:}B \\
\langle\!\langle G'\rangle\!\rangle & = & \mathsf{let}\, x_6 = (\lambda x{:}A.\mathsf{let}\, x_1 = x, x_2 = x, x_4 = (x_3x_2), x_5 = (x_4x_1)\,\mathsf{in}\, x_5)\,\mathsf{in} \\
& & \mathsf{let}\, x_8 = (x_7x_6)\,\mathsf{in}\,\mathsf{let}\, x_9 = (x_6x_8)\,\mathsf{in}\,[-] \\
& & \textit{in context}\ x_3{:}A{\rightarrow}A{\rightarrow}B, x_7{:}(A{\rightarrow}B){\rightarrow}A; x_6{:}A{\rightarrow}B, x_8{:}A, x_9{:}B
\end{array}$$

Note that the let-environment Δ only plays a role in restricting the variables that we are allowed to "plug in" the context when forming a λ-abstraction.

4.4.2 Independence of the box-topological ordering

The translation to the $\lambda\!\rightarrow\!+\,\mathsf{let}$ calculus has been defined with respect to a box-topological ordering. Now we will show that the specific box-topological ordering matters only for the order in which mutual independent let-bindings appear in the context. We use the result about interchanging neighbouring elements of linear extensions (Chapter 2, Theorem 2.1.12).

Lemma 4.4.5 *Let* G *be a deduction graph and let* φ *and* ψ *be box-topological orderings of* G*. If* $\widehat{\varphi} = \widehat{\psi}$ *and* $\widehat{\varphi_{|\mathcal{B}^m}} = \widehat{\psi_{|\mathcal{B}^m}}$ *for every box* $\mathcal{B}$*, then*

$$\langle\!\langle G\rangle\!\rangle_\varphi = \langle\!\langle G\rangle\!\rangle_\psi.$$

Proof The proof is by induction to the depth of G and the number of top-level nodes.

- Suppose the depth of G is 0 (there are no boxes in G). Then:

$$\langle\!\langle G\rangle\!\rangle_\varphi = \langle\!\langle G\rangle\!\rangle_{\widehat{\varphi}} = \langle\!\langle G\rangle\!\rangle_{\widehat{\psi}} = \langle\!\langle G\rangle\!\rangle_\psi$$

- Suppose the depth of G is $n+1$ and we know that the statement holds for deduction graphs with a lower depth.

  Suppose G has just one node on top-level, say k. Then k is an l-node. So

$$\begin{array}{rcl}
\langle\!\langle G\rangle\!\rangle_\varphi & = & \langle\!\langle G \setminus \mathcal{B}\rangle\!\rangle_{\varphi_{|G\setminus\mathcal{B}}}[\mathsf{let}\, x_k = (\lambda x_m{:}A.\langle\!\langle \mathcal{B}^m\rangle\!\rangle_{\varphi_{|\mathcal{B}^m}}[x_j]\,\mathsf{in}\,[-]] \\
& = & \langle\!\langle G \setminus \mathcal{B}\rangle\!\rangle_{\psi_{|G\setminus\mathcal{B}}}[\mathsf{let}\, x_k = (\lambda x_m{:}A.\langle\!\langle \mathcal{B}^m\rangle\!\rangle_{\psi_{|\mathcal{B}^m}}[x_j]\,\mathsf{in}\,[-]] \\
& = & \langle\!\langle G\rangle\!\rangle_\psi
\end{array}$$

Suppose G has more than one top-level node. Then the statement follows by case-analysis to the type of the maximum node and by the induction hypothesis.

Lemma 4.4.6 *If G is a deduction graph and φ and $\varphi\circ\pi_i$ are both box-topological orderings of G for a certain i, then*

$$\langle\!\langle G\rangle\!\rangle_{\varphi} \simeq \langle\!\langle G\rangle\!\rangle_{\varphi\circ\pi_i}$$

Proof Let $\psi := \varphi \circ \pi_i$. Note that if $\widehat{\varphi} = \widehat{\psi}$ and $\varphi_{|\widehat{\mathcal{B}^m}} = \psi_{|\widehat{\mathcal{B}^m}}$ for every box $\mathcal{B}$, then it follows by Lemma 4.4.5. Furthermore, if $\widehat{\varphi} \neq \widehat{\psi}$, then there exists a j such that $\widehat{\psi} = \widehat{\varphi} \circ \pi_j$. Similarly, if there exists a box $\mathcal{B}$ such that $\varphi_{|\widehat{\mathcal{B}^m}} \neq \psi_{|\widehat{\mathcal{B}^m}}$, then there exists a j such that $\psi_{|\widehat{\mathcal{B}^m}} = \varphi_{|\widehat{\mathcal{B}}} \circ \pi_j$.

Suppose $\widehat{\varphi} \neq \widehat{\psi}$ and $\langle\!\langle \mathcal{B}^m\rangle\!\rangle_{\varphi_{|\mathcal{B}^m}} \simeq \langle\!\langle \mathcal{B}^m\rangle\!\rangle_{\psi_{|\mathcal{B}^m}}$ for all boxes $\mathcal{B}$. Determine j such that $\widehat{\psi} = \widehat{\varphi} \circ \pi_j$. Suppose neither of the nodes $\varphi(i)$ and $\varphi(i+1)$ is an A-node.

Suppose $\widehat{\varphi}(j+1)$ is the φ-maximal element of G. Let $\varphi' := \varphi_{|G\setminus\widehat{\varphi}(j+1)}$; let $\varphi'' := \varphi_{|G\setminus\widehat{\varphi}(j+1),\widehat{\varphi}(j)}$; let $\psi' := \psi_{|G\setminus\widehat{\psi}(j+1)}$; let $\psi'' := \psi_{|G\setminus\widehat{\psi}(j+1),\widehat{\psi}(j)}$. Then there exist terms P, Q such that

$$\begin{aligned}
\langle\!\langle G\rangle\!\rangle_{\varphi} &= \langle\!\langle G \setminus \widehat{\varphi}(j+1)\rangle\!\rangle_{\varphi'}[\mathsf{let}\, x_{\widehat{\varphi}(j+1)} = P\, \mathsf{in}\, [-]] && (4.1)\\
&= \langle\!\langle G \setminus \widehat{\varphi}(j+1), \widehat{\varphi}(j)\rangle\!\rangle_{\varphi''}[\mathsf{let}\, x_{\widehat{\varphi}(j)} = Q\, \mathsf{in}\, \mathsf{let}\, x_{\widehat{\varphi}(j+1)} = P\, \mathsf{in}\, [-]] && (4.2)\\
&\simeq \langle\!\langle G \setminus \widehat{\psi}(j+1), \widehat{\psi}(j)\rangle\!\rangle_{\psi''}[\mathsf{let}\, x_{\widehat{\psi}(j)} = P\, \mathsf{in}\, \mathsf{let}\, x_{\widehat{\psi}(j+1)} = Q\, \mathsf{in}\, [-]] && (4.3)\\
&= \langle\!\langle G\rangle\!\rangle_{\psi} && (4.4)
\end{aligned}$$

We conclude (4.3) after case-analysis of the types of the nodes $\varphi(j)$ and $\varphi(j+1)$, seeing that $x_{\widehat{\varphi}(j)} \notin Q$ and $x_{\widehat{\varphi}(j+1)} \notin P$. Hence $\langle\!\langle G\rangle\!\rangle_{\varphi} \simeq \langle\!\langle G\rangle\!\rangle_{\varphi\circ\pi_i}$. If $\varphi(i)$ or $\varphi(i+1)$ is an A-node, we can make a similar computation.

The statement follows by induction to the depth of G and the number of top-level nodes.

Lemma 4.4.7 *If G is a deduction graph and φ and ψ are both box-topological orderings of G, then*

$$\langle\!\langle G\rangle\!\rangle_{\varphi} \simeq \langle\!\langle G\rangle\!\rangle_{\psi}$$

Proof Immediate by Lemma 2.1.20, Lemma 2.1.12 and Lemma 4.4.6.

From now on we will often omit the box-topological ordering from the notation. So we will write $\langle\!\langle G\rangle\!\rangle$ instead of $\langle\!\langle G\rangle\!\rangle_{\varphi}$.

4.4.3 Preservation of Reduction

This section shows that if G and G' are deduction graphs and G' is obtained from G by repeat-elimination, unsharing, incorporation or safe cut-elimination, then $\langle\!\langle G\rangle\!\rangle \twoheadrightarrow^{+}_{\mathrm{let}} \langle\!\langle G'\rangle\!\rangle$.

Lemma 4.4.8 (Gluing Lemma) *Let G and F be deduction graphs, such that for all free nodes (n, A) of F there exists a node (n, A) in G on top-level. Let $H = G \cup F$ (so the free nodes of F and the corresponding nodes of G have been identified). Then*

$$\langle\!\langle H \rangle\!\rangle = \langle\!\langle G \rangle\!\rangle[\langle\!\langle F \rangle\!\rangle].$$

Proof The proof is by induction on the number of non-A-nodes of F. Suppose F has only A-nodes. Then

$$\langle\!\langle H \rangle\!\rangle = \langle\!\langle G \rangle\!\rangle = \langle\!\langle G \rangle\!\rangle[-] = \langle\!\langle G \rangle\!\rangle[\langle\!\langle F \rangle\!\rangle].$$

Suppose F has $k+1$ non-A-nodes. Then we can choose the maximal node of H to be one of them, say r. Then

$$\begin{array}{rcl} \langle\!\langle H \rangle\!\rangle & = & \langle\!\langle H \setminus r \rangle\!\rangle[\mathsf{let}\, x_r = P\, \mathsf{in}\, [-]] \\ & = & (\langle\!\langle G \rangle\!\rangle[\langle\!\langle F \setminus r \rangle\!\rangle])[\mathsf{let}\, x_r = P\, \mathsf{in}\, [-]] \\ & = & \langle\!\langle G \rangle\!\rangle([\langle\!\langle F \setminus r \rangle\!\rangle][\mathsf{let}\, x_r = P\, \mathsf{in}\, [-]]) \\ & = & \langle\!\langle G \rangle\!\rangle[\langle\!\langle F \rangle\!\rangle] \end{array}$$

Lemma 4.4.9 *Let G be a deduction graph with free node (m, A). Let G' be G with (m, A) replaced by (n, A). Then*

$$\langle\!\langle G' \rangle\!\rangle = \langle\!\langle G \rangle\!\rangle[x_m := x_n].$$

Proof The proof is by induction on the construction of $\langle\!\langle G \rangle\!\rangle$.

Lemma 4.4.10
If G' is obtained from G by eliminating a safe cut, then $\langle\!\langle G \rangle\!\rangle \twoheadrightarrow_B \langle\!\langle G' \rangle\!\rangle$.

Proof Suppose p is top-level and maximal. We will proceed by cutting both G and G' in three pieces. The pieces of G are:

- $\langle \{p, n, m\}, \{(p, n), (p, m)\} \rangle$ (the nodes p, n, m with the edges between them);
- box $\mathcal{B}$ with the nodes that are reachable from within $\mathcal{B}$ in one step;
- $G \setminus \mathcal{B}, p$.

The pieces of G' are:

- $\langle \{p, j\}, \{(p, j)\} \rangle$ (the nodes p and j with an edge from p to j);
- $\mathcal{B}^k$ with k replaced by m;
- $G \setminus \mathcal{B}, p$.

Using the Lemmas 4.4.8 and 4.4.9, we see that:

$$\langle\!\langle G \rangle\!\rangle = \langle\!\langle G \setminus \mathcal{B}, p \rangle\!\rangle[\mathsf{let}\, x_n = (\lambda x_k{:}A.\langle\!\langle B^k \rangle\!\rangle[x_j])\, \mathsf{in}\, \mathsf{let}\, x_p = (x_n x_m)\, \mathsf{in}\, [-]]$$

and

$$\langle\!\langle G' \rangle\!\rangle = \langle\!\langle G \setminus \mathcal{B}, p \rangle\!\rangle[\mathcal{B}^k[x_k := x_m][\mathsf{let}\, x_p = x_j\, \mathsf{in}\, [-]].$$

Thus $\langle\!\langle G \rangle\!\rangle \twoheadrightarrow_B \langle\!\langle G' \rangle\!\rangle$. The other cases follow by induction. Here we make use of the facts that n and p are at the same level and that there exists just one edge to n.

Lemma 4.4.11
If G' is obtained from G by an incorporation step, then $\langle\!\langle G \rangle\!\rangle \twoheadrightarrow_{CM} \langle\!\langle G' \rangle\!\rangle$.

Proof Suppose n is a top-level node. Suppose box $\mathcal{B}_q$ with box-node q that contains p is a maximal element of G. Then

$$\begin{array}{rcl} \langle\!\langle G \rangle\!\rangle & = & \langle\!\langle G \setminus q \rangle\!\rangle[\mathsf{let}\, x_q = (\lambda x_k.\langle\!\langle \mathcal{B}_q^k \rangle\!\rangle[y])\, \mathsf{in}\, [-]] \\ & = & \langle\!\langle G \setminus q, n \rangle\!\rangle[\mathsf{let}\, x_n = (\lambda x_l.\langle\!\langle \mathcal{B}_n^l \rangle\!\rangle[x_j])\, \mathsf{in}\, \mathsf{let}\, x_q = (\lambda x_k.\langle\!\langle \mathcal{B}_q^k \rangle\!\rangle[y])\, \mathsf{in}\, [-]] \end{array}$$

The deduction graph G' contains now a box $\mathcal{C}_q$ that incorporates the box $\mathcal{B}_n$. To calculate $\langle\!\langle G' \rangle\!\rangle$, we divide $\mathcal{C}_q^k$ into two pieces:

- $\mathcal{B}_n$ together with all free nodes of $\mathcal{C}_q^k$;
- $\mathcal{C}_q^k \setminus \mathcal{B}_n$ (but with a free node n).

Now with the help of Lemma 4.4.8 we see that:

$$\begin{array}{rcl} \langle\!\langle G' \rangle\!\rangle & = & \langle\!\langle G' \setminus q \rangle\!\rangle[\mathsf{let}\, x_q = (\lambda x_k.\langle\!\langle \mathcal{C}_q^k \rangle\!\rangle[y])\, \mathsf{in}\, [-]] \\ & = & \langle\!\langle G' \setminus q \rangle\!\rangle[\mathsf{let}\, x_q = (\lambda x_k.(\mathsf{let}\, x_n = (\lambda x_l.\langle\!\langle \mathcal{B}_n^l \rangle\!\rangle[x_j])\, \mathsf{in}\, [-])[\langle\!\langle \mathcal{C}_q^k \setminus \mathcal{B}_n \rangle\!\rangle][y])\, \mathsf{in}\, [-]] \\ & = & \langle\!\langle G \setminus q, n \rangle\!\rangle[\mathsf{let}\, x_q = (\lambda x_k.(\mathsf{let}\, x_n = (\lambda x_l.\langle\!\langle \mathcal{B}_n^l \rangle\!\rangle[x_j])\, \mathsf{in}\, \langle\!\langle \mathcal{B}_q^k \rangle\!\rangle[y])\, \mathsf{in}\, [-]]. \end{array}$$

So $\langle\!\langle G \rangle\!\rangle \twoheadrightarrow_{CM} \langle\!\langle G' \rangle\!\rangle$. The other cases follow by induction.

Lemma 4.4.12
If G' is obtained from G by an unsharing step, then $\langle\!\langle G \rangle\!\rangle \twoheadrightarrow_{CP} \langle\!\langle G' \rangle\!\rangle$.

Proof Suppose box $\mathcal{B}$ with box-node n is at top-level. We will cut G' in four parts:

- H', i.e. all nodes from which n or n' is reachable, all boxes that contain these nodes, plus the nodes that can be reached from these in one step;
- box $\mathcal{B}$ with the nodes that are reachable from within $\mathcal{B}$ in one step, plus the free nodes of H';
- box $\mathcal{B}'$ with the nodes that are reachable from within $\mathcal{B}'$ in one step, plus the free nodes of $H' \cup \mathcal{B}$;
- $(G' \setminus H, \mathcal{B}', \mathcal{B})$, together with all free nodes of the previous $H' \cup \mathcal{B}' \cup \mathcal{B}$.

With Lemma 4.4.8 we see that:

$$\langle\!\langle G' \rangle\!\rangle = \langle\!\langle G' \setminus H, \mathcal{B}', \mathcal{B} \rangle\!\rangle[\mathsf{let}\, x_{n'} = (\lambda x_m.\langle\!\langle \mathcal{B}^m \rangle\!\rangle[x_{j'}])\, \mathsf{in}\, \mathsf{let}\, x_n = (\lambda x_m.\langle\!\langle \mathcal{B}^m \rangle\!\rangle[x_j])\, \mathsf{in}\, \langle\!\langle H' \rangle\!\rangle]$$

We cut G in three parts as follows:

- H, i.e. all nodes from which n is reachable, all boxes that contain these nodes, plus the nodes that can be reached from these within one step;
- box $\mathcal{B}$ with the nodes that are reachable from within $\mathcal{B}$ in one step, plus the free nodes of H;
- $(G \setminus H, \mathcal{B})$, together with the free nodes of $H \cup \mathcal{B}$.

With Lemmas 4.4.8 and 4.4.9 we see that:

$$\begin{array}{rcl} \langle\!\langle G \rangle\!\rangle & = & \langle\!\langle G \setminus H, \mathcal{B} \rangle\!\rangle[\mathsf{let}\, x_n = (\lambda x_m.\langle\!\langle \mathcal{B}^m \rangle\!\rangle[x_j])\, \mathsf{in}\, \langle\!\langle H \rangle\!\rangle] \\ & = & \langle\!\langle G' \setminus H, \mathcal{B}', \mathcal{B} \rangle\!\rangle[\mathsf{let}\, x_n = (\lambda x_m.\langle\!\langle \mathcal{B}^m \rangle\!\rangle[x_j])\, \mathsf{in}\, \langle\!\langle H' \rangle\!\rangle[x_{n'} := x_n]] \end{array}$$

So $\langle\!\langle G \rangle\!\rangle \twoheadrightarrow_{CP} \langle\!\langle G' \rangle\!\rangle$. The other case follows by induction.

Lemma 4.4.13 *If G' is obtained from G by a repeat-elimination step, then $\langle\!\langle G \rangle\!\rangle \twoheadrightarrow_L \langle\!\langle G' \rangle\!\rangle$.*

Proof Suppose n_0 and n_1 are top-level nodes ($n_0 \longrightarrow\!\triangleright\, n_1$). By cutting G in three parts and G' in either two or three, we easily see that $\langle\!\langle G \rangle\!\rangle \twoheadrightarrow_L \langle\!\langle G' \rangle\!\rangle$. Now assume that n_1 is in a box. Using that none of the nodes to n_1 is an I-node (because then that would be the only edge to n_1, and we know that n_1 is part of a cut), we can make another deduction graph, G'' as follows. Place a node n_1' at top level. Replace the edges $n_1 \longrightarrow\!\triangleright\, n_0$ by $n_1' \longrightarrow\!\triangleright\, n_0$ and redirect all edges to n_1 to n_1'. We can now remove n_1. We have that G' can be obtained from G'' by a repeat-elimination step, and because both n_0 and n_1' are at top-level, $\langle\!\langle G'' \rangle\!\rangle \twoheadrightarrow_L \langle\!\langle G' \rangle\!\rangle$. G, on the other hand, can be obtained from G'' by an incorporation step (this time on a single node, instead of on a box), and a renaming. Thus $\langle\!\langle G'' \rangle\!\rangle$, $\langle\!\langle G' \rangle\!\rangle$ and $\langle\!\langle G \rangle\!\rangle$ have the following form, where $x_j \neq x_{n_1'}$:

$$\langle\!\langle G'' \rangle\!\rangle = \langle\!\langle G'' \setminus n_1', \mathcal{B} \rangle\!\rangle[\mathsf{let}\, x_{n_1'} = x_{n_0}\, \mathsf{in}\, \mathsf{let}\, x_r = (\lambda x_k.\langle\!\langle \mathcal{B}^k \rangle\!\rangle[x_j])\, \mathsf{in}\, [-]]$$

$$\langle\!\langle G' \rangle\!\rangle = \langle\!\langle G'' \setminus n_1', \mathcal{B} \rangle\!\rangle[\mathsf{let}\, x_r = (\lambda x_k.\langle\!\langle \mathcal{B}^k \rangle\!\rangle[x_j])\, \mathsf{in}\, [-]][x_{n_1'} := x_{n_0}]$$

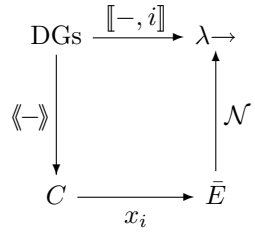
$$\langle\!\langle G \rangle\!\rangle = \langle\!\langle G'' \setminus n_1', \mathcal{B} \rangle\!\rangle[\mathsf{let}\, x_r = (\lambda x_k.\mathsf{let}\, x_{n_1'} = x_{n_0}\, \mathsf{in}\, \langle\!\langle \mathcal{B}^k \rangle\!\rangle)\, \mathsf{in}\, [-]]$$

So indeed $\langle\!\langle G \rangle\!\rangle \twoheadrightarrow_L \langle\!\langle G' \rangle\!\rangle$. Other cases follow by induction.

4.5 Connecting the Translations

We want to formally state the connection between $[\![-,-]\!]$ and $\langle\!\langle - \rangle\!\rangle$. That is, we want to prove that the following diagram commutes: for i a top-level node of a deduction graph G, we want to prove that $\mathcal{N}(\langle\!\langle G \rangle\!\rangle[x_i]) = [\![G_i, i]\!]$, where $\mathcal{N}$ is

the function that contracts let-redexes, as will be formally defined in Definition 4.5.7. The bottom arrow indicates the substitution of x_i in the context.



We will now make the ingredients of this diagram precise.

To capture both simply typed terms and terms of the form $C[x]$, where $\Gamma; \Delta \vdash C$ and $x \in \Gamma \cup \Delta$, we consider the following syntactic domain of λ-let-expressions:

$$\bar{E} \quad := \quad x \mid (\bar{E}\bar{E}) \mid \mathsf{let}\, x = \bar{E}\, \mathsf{in}\, \bar{E} \mid \lambda x{:}A.\bar{E}$$

Definition 4.5.1 *We define the following typing rules for λ-let-terms:*

$$\frac{\Gamma \vdash_l T : A \;\; \Gamma, x{:}A \vdash_l S{:}B}{\Gamma \vdash_l \mathsf{let}\, x = T\, \mathsf{in}\, S{:}B}\, \mathsf{let}_l\text{-rule} \qquad \Gamma \vdash_l x : A \quad \text{if } (x{:}A) \in \Gamma$$

$$\frac{\Gamma, x{:}A \vdash_l T}{\Gamma \vdash_l \lambda x{:}A.T : A{\rightarrow}B}\, \lambda_l\text{-rule} \qquad \frac{\Gamma \vdash_l T : A{\rightarrow}B \;\; \Gamma \vdash_l S : A}{\Gamma \vdash_l TS : B}\, \mathit{app}_l\text{-rule}$$

Lemma 4.5.2

1. *If $\Gamma \vdash M{:}A$, then $\Gamma \vdash_l M{:}A$*
2. *If $\Gamma; \Delta \vdash C$ and $x{:}A \in \Gamma \cup \Delta$, then $\Gamma \vdash_l C[x]{:}A$*

Proof By mutual induction on the derivation.

- Suppose $\Delta = <>$ and $C = [-]$. For $x \in \Gamma \cup \Delta$: $\Gamma \vdash_l x{:}A$.
- Suppose $M = x$ and $x{:}A \in \Gamma$, then $\Gamma \vdash_l x{:}A$.
- Suppose $\Gamma \vdash x{:}A{\rightarrow}B$ and $\Gamma \vdash y{:}A$, then by induction $\Gamma \vdash_l x{:}A{\rightarrow}B$ and $\Gamma \vdash_l y{:}A$. By the app_l-rule we get: $\Gamma \vdash_l xy{:}B$.
- Suppose $\Gamma \vdash T{:}A$ and $\Gamma, x{:}A; \Delta \vdash C$ and $y{:}B \in \Gamma \cup \Delta, x{:}A$. Then by induction $\Gamma \vdash_l T{:}A$ and $\Gamma, x{:}A \vdash_l C[y]{:}B$. By the let_l-rule we get: $\Gamma \vdash_l \mathsf{let}\, x = T\, \mathsf{in}\, C[y]{:}B$.
- Suppose $\Gamma, x{:}A; \Delta \vdash C$ and $y{:}B \in \Delta$. Then by induction $\Gamma, x{:}A \vdash_l C[y]{:}B$. So, by the λ_l-rule $\Gamma \vdash_l \lambda x{:}A.C[y]{:}B$

Definition 4.5.3 *We define the reduction rule $\twoheadrightarrow_{l\mathsf{let}}$ as the contextual closure of the following rewrite rule on λ-let-terms:*

$$\mathsf{let}\, x = T\, \mathsf{in}\, S \twoheadrightarrow_{l\mathsf{let}} S[x := T]$$

Lemma 4.5.4 (Substitution) *If* $\Gamma, x{:}A \vdash_l T{:}B$ *and* $\Gamma \vdash_l S{:}A$, *then* $\Gamma \vdash_l T[x := S]$.

Proof By induction on the construction of T.

Lemma 4.5.5 (Subject Reduction) *If* $\Gamma \vdash_l T$ *and* $T \twoheadrightarrow_{l\mathsf{let}} T'$, *then* $\Gamma \vdash_l T'$.

Proof Suppose $\Gamma \vdash_l \mathsf{let}\, x = T \,\mathsf{in}\, S{:}B$. Then $\Gamma \vdash_l T{:}A$ for some A and $\Gamma, x{:}A \vdash_l S{:}B$. Hence by Substitution: $\Gamma \vdash_l T[x := S]$.

Lemma 4.5.6 *The reduction* $\twoheadrightarrow_{l\mathsf{let}}$ *is strongly normalising and confluent.*

Proof We make a map I, from λ-let-terms to terms of simply typed λ-calculus as follows: $I(\mathsf{let}\, x = T \,\mathsf{in}\, S) = (\lambda x{:}A.I(S))\ I(T)$, and I is the identity on all other terms. Then: if $T \twoheadrightarrow_{l\mathsf{let}} T'$, then $I(T) \twoheadrightarrow_\beta I(T')$. Because $\twoheadrightarrow_\beta$-reduction is strongly-normalising, $\twoheadrightarrow_{l\mathsf{let}}$-reduction is too. Note that $\twoheadrightarrow_{l\mathsf{let}}$ is weakly confluent, and we are done.

Definition 4.5.7 *We will write* $\mathcal{N}(T)$ *for the* $l\mathsf{let}$*-normal form of* T.

Let C be a well-formed context. We use the following properties:

1. If $(C[x_n\ x_m])$ is a well-typed λ-let-term, then
$$\mathcal{N}(C[x_n\ x_m]) = (\mathcal{N}(C[x_n])\ \mathcal{N}(C[x_m]))$$
2. If $(C[\lambda x.D])$ is a well-typed λ-let-term, then
$$\mathcal{N}(C[\lambda x.D]) = \lambda x.\mathcal{N}(C[D])$$

Definition 4.5.8 *Let* G *be a deduction graph with free nodes* $(n_1, A), \ldots, (n_k, A)$. $G^{m,n_1,\ldots,n_k}$ *is the graph consisting of* G, *a new node* (m, A), *and edges* $n_1 \longrightarrow\!\triangleright m, \ldots, n_k \longrightarrow\!\triangleright m$.

Lemma 4.5.9 *Let* G *be a deduction graph with free nodes* $(n_1, A), \ldots, (n_k, A)$. *Then* $G^{m,n_1,\ldots,n_k}$ *is a deduction graph.*

Proof By Lemma 2.2.7.

Lemma 4.5.10 *Let* G *be a deduction graph. If* $(n, A{\to}B)$ *is a box-node with* $(n, A{\to}B) \longrightarrow\!\triangleright (j, B)$ *and the* A*-nodes of the box are* $(n_1, A), \ldots, (n_k, A)$, *then*
$$[\![G, n]\!] = \lambda x_m{:}A.[\![G_j^{m,n_1,\ldots,n_k}, j]\!]$$

Proof By Lemma 4.2.7

Theorem 4.5.11 *Let* G *be a deduction graph and let* i *be a top-level node of* G. *Then:*
$$\mathcal{N}(\langle\!\langle G \rangle\!\rangle[x_i]) = [\![G_i, i]\!]$$

Proof Without loss of generality, we assume that i is a maximal node.

A Suppose i is an A-node. Then x_i is not let-bound in $\langle\!\langle G \rangle\!\rangle$. Hence $\mathcal{N}\langle\!\langle G \rangle\!\rangle[x_i] \equiv x_i \equiv [\![G_i, i]\!]$.

R Suppose i is an R-node, say $i \longrightarrow\!\triangleright n$ for some node n. Then:

$$\begin{array}{rcl} \mathcal{N}(\langle\!\langle G \rangle\!\rangle[x_i]) & = & \mathcal{N}(\langle\!\langle G \setminus i \rangle\!\rangle[\mathsf{let}\, x_i = x_n \,\mathsf{in}\, x_i]) \\ & = & \mathcal{N}(\langle\!\langle G \setminus i \rangle\!\rangle[x_n]) = [\![G_n, n]\!] = [\![G_i, i]\!] \end{array}$$

E Suppose i is an E-node, say $(i, B) \longrightarrow\!\triangleright (n, A{\rightarrow}B)$ and $(i, B){\rightarrow}(m, A)$. Then:

$$\begin{array}{rcl} \mathcal{N}(\langle\!\langle G \rangle\!\rangle[x_i]) & = & \mathcal{N}(\langle\!\langle G \setminus i \rangle\!\rangle[\mathsf{let}\, x_i = (x_n\ x_m)\,\mathsf{in}\, x_i]) \\ & = & \mathcal{N}(\langle\!\langle G \setminus i \rangle\!\rangle[(x_n\ x_m)]) = (\mathcal{N}(\langle\!\langle G \setminus i \rangle\!\rangle[x_n]\ \mathcal{N}(\langle\!\langle G \setminus i \rangle\!\rangle[x_m]) \\ & = & ([\![G_n, n]\!]\ [\![G_m, m]\!]) = [\![G_i, i]\!] \end{array}$$

I Suppose i is an I-node, say $(i, A{\rightarrow}B) \longrightarrow\!\triangleright (j, B)$, with associated box $\mathcal{B}$. Then

$$\begin{array}{rcll} \mathcal{N}(\langle\!\langle G \rangle\!\rangle[x_i]) & = & \mathcal{N}(\langle\!\langle G \setminus i \rangle\!\rangle[\mathsf{let}\, x_i = \lambda x_m{:}A.\langle\!\langle B^m \rangle\!\rangle[x_j]\,\mathsf{in}\, x_i]) & (4.5) \\ & = & \mathcal{N}(\langle\!\langle G \setminus i \rangle\!\rangle[\lambda x_m{:}A.\langle\!\langle B^m \rangle\!\rangle[x_j]]) & (4.6) \\ & = & \lambda x_m{:}A.\mathcal{N}(\langle\!\langle G \setminus i \rangle\!\rangle[\langle\!\langle B^m \rangle\!\rangle[x_j]]) & (4.7) \\ & = & \lambda x_m{:}A.\mathcal{N}(\langle\!\langle G^{m,y_1,\ldots,y_k} \setminus i \rangle\!\rangle[x_j]) & (4.8) \\ & = & \lambda x_m{:}A.[\![(G^{m,y_1,\ldots,y_k} \setminus i)_j, j]\!] & (4.9) \\ & = & [\![G_i, i]\!] & (4.10) \end{array}$$

We use Lemma 4.4.8 to go from (4.7) to (4.8). We use Lemma 4.5.10 to go from (4.9) to (4.10).

4.6 Other Explicit Substitution Calculi

There are many well-known explicit substitution calculi and let-calculi, like the ones found in [64], [59], [13], and [46]. The question therefore arises whether the reductions of the $\lambda{\rightarrow} + \mathsf{let}$-calculus can be composed out of a "contextual variant" of one of those. In this section we take the $\lambda\mathsf{x}c^-$-calculus [13] as a generic example to support the claim that that is not the case.

Definition 4.6.1 *The set of terms of $\lambda\mathsf{x}c^-$ is defined as follows:*

$$A ::= x \mid AA \mid \lambda x.A \mid A\langle x = A\rangle$$

Definition 4.6.2 *The* x-reduction rules *on $\lambda\mathsf{x}c^-$ are the contextual closures of the following rules:*

$$\begin{array}{rcl} (AB)\langle x = C\rangle & \longrightarrow_{\mathsf{x}App} & (A\langle x = C\rangle)(B\langle x = C\rangle) \\ (\lambda y.A)\langle x = C\rangle & \longrightarrow_{\mathsf{x}Abst} & \lambda y.A\langle x = C\rangle \ \textit{ if } x \not\equiv y \textit{ and } y \notin FV(C) \\ x\langle x = C\rangle & \longrightarrow_{\mathsf{x}Var} & C \\ A\langle x = C\rangle & \longrightarrow_{\mathsf{x}Gar} & A \ \textit{ if } x \notin FV(A) \end{array}$$

where A, B, C are $\lambda\mathsf{x}c^-$-terms. We will write $\twoheadrightarrow_{\mathsf{x}}$ for the union of these rules.

Lemma 4.6.3 *The reduction $\twoheadrightarrow_{\mathsf{x}}$ is strongly normalising and confluent.*

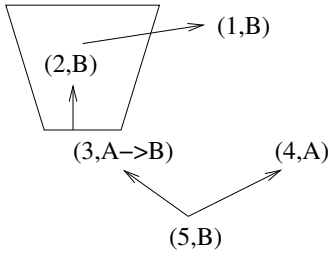
Proof See [13].

We will write $\mathsf{x}(A)$ for the $\twoheadrightarrow_{\mathsf{x}}$ normal form of A.

Definition 4.6.4 *The* β-reduction rule *and the* c^--reduction rule *are the contextual closure of the following rules respectively:*

$$\begin{array}{rcl} (\lambda x.A)B & \twoheadrightarrow_{\beta} & A\langle x = B\rangle \\ A\langle x = B\rangle\langle y = C\rangle & \twoheadrightarrow_{c^-} & A\langle x = B\langle y = C\rangle\rangle \;\; \textit{if } x \in FV(\mathsf{x}(A)),\, y \notin FV(\mathsf{x}(A)) \end{array}$$

Consider the following simple deduction graph G:



G has just one node without incoming edges, which enables us to compare the $\lambda\!\rightarrow\! +\, \mathsf{let}$-*context* with an $\lambda\mathsf{x}c^-$-*term*, simply by putting x_5 in the hole of the context. We will show that when we consider the $\lambda\!\rightarrow\! +\, \mathsf{let}$-context of G as an $\lambda\mathsf{x}c^-$-term and we apply $\lambda\mathsf{x}c^-$-reduction rules to it, we will not get the term that we would associate to G', the graph obtained by eliminating the safe cut of G. This has nothing to do with G' being disconnected, as 4 is a free node anyway, and does not contribute to the $\lambda\!\rightarrow\! +\, \mathsf{let}$-context. We could have made a similar example in which the graph after cut-elimination is still connected, but that would have made the example harder to read.

We see that $\langle\!\langle G \rangle\!\rangle =$

$$\mathsf{let}\, x_3 = (\lambda y.\mathsf{let}\, x_2 = x_1\, \mathsf{in}\, x_2)\, \mathsf{in}\, \mathsf{let}\, x_5 = (x_3 x_4)\, \mathsf{in}\, [-]$$

So in $\lambda\mathsf{x}c^-$-notation and with x_5 put into the hole, that is:

$$x_5\langle x_5 = (x_3 x_4)\rangle\langle x_3 = \lambda y.x_2\langle x_2 = x_1\rangle\rangle$$

And $\langle\!\langle G' \rangle\!\rangle =$

$$\mathsf{let}\, x_2 = x_1\, \mathsf{in}\, \mathsf{let}\, x_5 = x_2\, \mathsf{in}\, [-]$$

Which corresponds to:

$$x_5\langle x_5 = x_2\rangle\langle x_2 = x_1\rangle$$

We now show that $x_5\langle x_5 = (x_3 x_4)\rangle\langle x_3 = \lambda y.x_2\langle x_2 = x_1\rangle\rangle$ does not reduce to $x_5\langle x_5 = x_2\rangle\langle x_2 = x_1\rangle$ in the $\lambda\mathsf{x}c^-$-calculus.

The most promising reduction path is the following. First we apply $\Longrightarrow_{c^-}$ which results in:

$$x_5\langle x_5 = (x_3x_4)\langle x_3 = \lambda y.x_2\langle x_2 = x_1\rangle\rangle\rangle$$

Then we do a $\Longrightarrow_{\mathtt{x}App}$-reduction, a $\Longrightarrow_{\mathtt{x}Gar}$-reduction and a $\Longrightarrow_{\mathtt{x}Var}$-reduction, and we continue as follows:

$$\begin{array}{lll} x_5\langle x_5 = (\lambda y.x_2\langle x_2 = x_1\rangle)x_4)\rangle & \Longrightarrow_{\beta} & x_5\langle x_5 = x_2\langle x_2 = x_1\rangle\langle y = x_4\rangle\rangle \\ & \Longrightarrow_{\mathtt{x}Gar} & x_5\langle x_5 = x_2\langle x_2 = x_1\rangle\rangle \end{array}$$

This will not lead to $x_5\langle x_5 = x_2\rangle\langle x_2 = x_1\rangle$, as that would require $\Longrightarrow_{c^-}$ *from right to left.*

A similar problem restrains us from defining the $\lambda\!\!\rightarrow$ + let-calculus in a contextual variant of most other well-known substitution-calculi or let-calculi, with as possible exception the let-calculi introduced in the framework of linear and light logic, like [84]. The connection between these calculi and the $\lambda\!\!\rightarrow$ + let-calculus has yet to be explored.

Chapter 5

Relation with Proof Nets

5.1 Introduction

Deduction graphs (Chapter 2) provide a formalism for natural deduction where the deductions have the structure of an *acyclic directed graph* with *boxes*. The main advantage of this formalism is the reuse of sub-proofs, which is simply done by putting several arrows to its conclusion. Because we do not see this as a logical step, we call this kind of sharing *implicit*. This makes deduction graphs a generalization of both Gentzen-Prawitz style natural deduction and Fitch style flag deduction. Because of the graph structure, one has to be explicit about local assumptions, so the boxes are used to limit the scope of an assumption.

In proof nets for multiplicative exponential linear logic (MELL) [36], the re-use of formulas is done by contraction links, and is therefore *explicit*. Proof nets also have boxes. Here they sequentialise the proof net in order to introduce the global modality "!" (of course). There is a process on the proof nets of MELL that removes cut-links, also called cut-elimination.

There are some well-known translations from simply typed λ-terms or natural deduction derivations to proof nets [36]. There are three reasons why a translation from deduction graphs to proof nets is more difficult:

1. Parts of a deduction graph can be shared. We want a translation to somehow reflect this sharing. Simply typed λ-terms do not have sharing, except for the variables.

2. Deduction graphs contain boxes. We would like a translation to associate a proof net box to a deduction graph box.

3. The sharing and the boxes are both affected during the process of cut-elimination on deduction graphs. We want the cut-elimination on deduction graphs to behave similarly to cut-elimination on proof nets (on the translated deduction graphs).

4. Deduction graphs may have many conclusions, unlike simply typed terms, which have just one type.

In this chapter we present two translations from deduction graphs to proof nets, which mainly differ in the way they handle multiple conclusions.

We start with a brief overview of linear logic and proof nets of MELL, and the transformations involved in cut-elimination of proof-nets (Section 5.2). For an extensive treatment of linear logic, see [87]

Then we present *deduction graphs with explicit sharing*, `SG`, which is a variant of the usual definition of deduction graphs. The main difference is that in `SG`, sharing is handled as a logical step (Section 5.3). Furthermore we add a step, "loop", that allows to distinguish between conclusions and garbage, that is between formulas that we can use as the premises of next steps and formulas that will not be used any more.

The translation $(-)^*$ of Girard is used to translate formulas of minimal proposition logic to formulas of MELL (Section 5.4). This translation is the most suitable to our needs because it allows to nicely map deduction graph boxes to proof net boxes.

The direct translation (Section 5.5) uses *tensors* ($\otimes$) to connect the various conclusions. Just as most translations from λ-calculus to proof nets, assumptions Γ occur in the translation as terminal nodes $\Gamma^{*\perp}$, so as the *negation* of the translated formulas. An advantage of the direct translation is, that it only uses concepts that are already known from the theory of proof nets. A disadvantage is, that it works from conclusions to assumptions. So to translate a deduction graph, we must know that the construction of the graph has been finished. If we would decide to extend the graph later, we would have to translate the whole graph again.

Context nets (Section 5.6) form an extension of the concept of proof nets. They have terminal nodes, but they also permit so called *initial nodes*. We focus on *closed one-liners*, a sub-class of context nets and we study some closure properties of this class.

We argue that there are several sensible translations from deduction graphs to proof nets, the translation of Section 5.5 being one of them, and that the translation to context nets (Section 5.7) generalises them by investigating the common parts of those proof nets. An other benefit is that the translation from deduction graphs to context nets treats the assumptions and conclusions of deduction graphs in a uniform way: Not only do the assumptions Γ associate to terminal nodes $\Gamma^{*\perp}$, but the conclusions Δ associate to initial nodes $\Delta^{*\perp}$ as well. Moreover, the translation can be done in the same order as the construction of the deduction graph, allowing to work with graphs that have not yet been completed. In fact, the construction of the translation of a deduction graph looks like the construction of the original deduction graph "up side down".

In Section 5.8 it is shown that the translation via context nets, after a slight modification, yields the same results as the direct translation. Section 5.9 shows that the translations behave well with respect to cut-elimination. Finally, Section 5.10 discusses various strong normalisation results that can be concluded from Section 5.9, by interpreting the transformation rules in a stricter or more liberal sense.

5.2 Linear Logic and Proof Nets

5.2.1 Linear Logic

Classical sequent calculus

Gentzen introduced in [27] the formalism of sequent calculus as a device to study natural deduction. A sequent is of the form $\Delta \vdash \Sigma$, where Δ and Σ are finite sequences of propositions. The sequent $\Delta \vdash \Sigma$ can be understood as "there exists a natural deduction with conclusions $\bigvee \Sigma$ (i.e. the disjunction of the formulas in Σ) from the assumption $\bigwedge \Delta$ (i.e. the conjunction of the formulas of Δ)".

Sequent calculus does not have elimination rules and introduction rules like natural deduction, but it contains left-introduction rules and right-introduction rules instead. The other rules of the calculus are the axiom-rule (A), the cut-rule (Cut), and the structural rules contraction (C_L and C_R), weakening (W_L and W_R), and permutation (P_L and P_R). Figure 5.1 shows the rules of the sequent calculus. The Greek letters range over finite sequences of propositions, whereas the Latin letters range over the propositions themselves.

Note that there are many choices of how to exactly formulate the rules. The right conjunction introduction, for example, could as well be formulated as:

$$\frac{\Delta \vdash A, \Sigma \quad \Delta \vdash B, \Sigma}{\Delta \vdash A \wedge B, \Sigma} \wedge_R$$

It is easy to see that these two variations of right conjunction introduction can be proven from each other using weakening and contraction.

Except the cut-rule, all rules have the *subformula-property*: all formulas in the upper sequent or upper sequents appear as (sub)formulas in the lower sequent. By proving that the cut-rule can be omitted, Gentzen showed that the sequent calculus is decidable, and hence he concluded that natural deduction is decidable as well.

Linear logic sequent calculus

Girard [36] came up with the sequent calculus for linear logic by reconsidering the rules for weakening and contraction, or, more general, by looking for a *resource-sensitive* logic.

Defining a system without weakening and contraction leaves one with the problem of which variation of the remaining rules to take: Variations that are equivalent in the system of classical sequent calculus, may not be equivalent any more without weakening and contraction. In practice this means that, for example, the right-conjunction rule does not lead to one right-conjunction rule in the calculus without weakening and contraction, but it splits in two different rules instead, corresponding to equivalent variations in the classical sequent calculus:

$$\frac{\Delta \vdash A, \Sigma \quad \Delta' \vdash B, \Sigma'}{\Delta, \Delta' \vdash A \otimes B, \Sigma, \Sigma'} \otimes_R \qquad \frac{\Delta \vdash A, \Sigma \quad \Delta \vdash B, \Sigma}{\Delta \vdash A \& B, \Sigma} \&_R$$

$$\frac{}{A \vdash A}\,\mathsf{A} \qquad \frac{\Sigma \vdash A, \Delta \quad \Sigma', A \vdash \Delta'}{\Sigma, \Sigma' \vdash \Delta, \Delta'}\,\mathsf{Cut}$$

$$\frac{\Delta \vdash A, \Sigma}{\Delta, \neg A \vdash \Sigma}\,\neg_L \qquad \frac{\Delta, A \vdash \Sigma}{\Delta \vdash \neg A, \Sigma}\,\neg_R$$

$$\frac{\Delta, A, B \vdash \Sigma}{\Delta, A \wedge B \vdash \Sigma}\,\wedge_L \qquad \frac{\Delta \vdash A, B, \Sigma}{\Delta \vdash A \vee B}\,\vee_R$$

$$\frac{\Delta \vdash A, \Sigma \quad \Delta' \vdash B, \Sigma'}{\Delta, \Delta' \vdash A \wedge B, \Sigma, \Sigma'}\,\wedge_R \qquad \frac{\Delta, A \vdash \Sigma \quad \Delta', B \vdash \Sigma'}{\Delta, \Delta', A \vee B \vdash \Sigma, \Sigma'}\,\vee_L$$

$$\frac{A, A, \Delta \vdash \Delta'}{A, \Delta \vdash \Delta'}\,\mathsf{C}_L \qquad \frac{\Delta \vdash A, A, \Delta'}{\Delta \vdash A, \Delta'}\,\mathsf{C}_R$$

$$\frac{\Delta \vdash \Sigma}{\Delta, A \vdash \Sigma}\,\mathsf{W}_L \qquad \frac{\Delta \vdash \Sigma}{\Delta \vdash A, \Sigma}\,\mathsf{W}_R$$

$$\frac{\Delta, A, B, \Delta' \vdash \Sigma}{\Delta, B, A, \Delta' \vdash \Sigma}\,\mathsf{P}_L \qquad \frac{\Delta \vdash \Sigma, A, B, \Sigma'}{\Delta \vdash \Sigma, B, A, \Sigma'}\,\mathsf{P}_R$$

Figure 5.1: Rules of the classical sequent calculus.

In the rest of our story, only the variations as presented in Figure 5.1 play a role, leading to the so-called *multiplicative* part of linear logic. However, this explains the need for new symbols, as an alternative for $\wedge$ and $\vee$.

As a next step, weakening and contraction are reintroduced, but this time as logical rules: Two modalities, the *exponentials* ? (*why not*) and ! (*of course*) take care of a more subtle application of these rules.

We now have all ingredients to formally define the formulas of linear logic proposition calculus.

Definition 5.2.1 *The formulas of linear logic proposition calculus are given by the following grammar:*

$$\begin{array}{lcl} \mathcal{A} & ::= & A_0 \mid A_1 \mid A_2 \mid \ldots \\ \mathcal{F} & ::= & \mathcal{A} \mid \neg\mathcal{F} \mid \mathcal{F} \otimes \mathcal{F} \mid \mathcal{F} \parr \mathcal{F} \mid !\mathcal{F} \mid ?\mathcal{F} \end{array}$$

The rules of the sequent calculus for multiplicative exponential linear logic (MELL) is presented in Figure 5.2. The Greek letters denote again finite sequences of formulas. By $!\Delta$ ($?\Sigma$) we mean that all formulas in Δ (in Σ) are of the form $!A$ ($?A$).

$$\frac{}{A \vdash A}\,\mathsf{A} \qquad \frac{\Sigma \vdash A, \Delta \quad \Sigma', A \vdash \Delta'}{\Sigma, \Sigma' \vdash \Delta, \Delta'}\,\mathsf{Cut}$$

$$\frac{\Delta \vdash A, \Sigma}{\Delta, \neg A \vdash \Sigma}\,\neg_L \qquad \frac{\Delta, A \vdash \Sigma}{\Delta \vdash \neg A, \Sigma}\,\neg_R$$

$$\frac{A, \Delta \vdash \Sigma}{!A, \Delta \vdash \Sigma}\,!_L \qquad \frac{\Delta \vdash A, \Sigma}{\Delta \vdash ?A, \Sigma}\,?_R$$

$$\frac{!\Delta \vdash A, ?\Sigma}{!\Delta \vdash !A, ?\Sigma}\,!_R \qquad \frac{!\Delta, A \vdash ?\Sigma}{!\Delta, ?A \vdash ?\Sigma}\,?_L$$

$$\frac{\Delta, A, B \vdash \Sigma}{\Delta, A \otimes B \vdash \Sigma}\,\otimes_L \qquad \frac{\Delta \vdash A, B, \Sigma}{\Delta \vdash A \parr B}\,\parr_R$$

$$\frac{\Delta \vdash A, \Sigma \quad \Delta' \vdash B, \Sigma'}{\Delta, \Delta' \vdash A \otimes B, \Sigma, \Sigma'}\,\otimes_R \qquad \frac{\Delta, A \vdash \Sigma \quad \Delta', B \vdash \Sigma'}{\Delta, \Delta', A \parr B \vdash \Sigma, \Sigma'}\,\parr_L$$

$$\frac{!A, !A, \Delta \vdash \Sigma}{!A, \Delta \vdash \Sigma}\,\mathsf{C}_L \qquad \frac{\Delta \vdash ?A, ?A, \Sigma}{\Delta \vdash ?A, \Sigma}\,\mathsf{C}_R$$

$$\frac{\Delta \vdash \Sigma}{\Delta, !A \vdash \Sigma}\,\mathsf{W}_L \qquad \frac{\Delta \vdash \Sigma}{\Delta \vdash ?A, \Sigma}\,\mathrm{W}_R$$

$$\frac{\Delta, A, B, \Delta' \vdash \Sigma}{\Delta, B, A, \Delta' \vdash \Sigma}\,\mathrm{P}_L \qquad \frac{\Delta \vdash \Sigma, A, B, \Sigma'}{\Delta \vdash \Sigma, B, A, \Sigma'}\,\mathrm{P}_R$$

Figure 5.2: Rules of the MELL sequent calculus.

One-sided linear logic

By dropping the negation as logical operator and introducing an involutive function $\perp$ on the formulas, called *linear negation*, the need for the turn style disappears. Moreover, The number of rules can be diminished, because some of them can be identified. This will be explained below.

The intended meaning of the linear negation is, that the negated formulas are the formulas on the left of the turn style. So basically, we obtain the *right*-sided logic. For example, the rule

$$\frac{\Delta \vdash A, \Sigma \quad \Delta' \vdash B, \Sigma'}{\Delta, \Delta' \vdash A \otimes B, \Sigma, \Sigma'} \quad \text{becomes} \quad \frac{\Delta^{\perp}, A, \Sigma \quad \Delta'^{\perp}, B, \Sigma'}{\Delta^{\perp}, \Delta'^{\perp}, A \otimes B, \Sigma, \Sigma'}$$

and the rule

$$\frac{\Delta, A \vdash \Sigma \quad \Delta', B \vdash \Sigma'}{\Delta, \Delta', A \parr B \vdash \Sigma, \Sigma'} \quad \text{becomes} \quad \frac{\Delta^\perp, A^\perp, \Sigma \quad \Delta'^\perp, B^\perp, \Sigma'}{\Delta^\perp, \Delta'^\perp, (A \parr B)^\perp, \Sigma, \Sigma'}.$$

So, when we identify $(A \parr B)^\perp$ with $A^\perp \otimes B^\perp$, we can catch these two rules in one.

Formally, we assume a function on the atoms, $\perp : \mathcal{A} \to \mathcal{A}$, and postulate that $A_i^{\perp\perp} = A_i$ for every $i \in \mathbb{N}$. Next we extend the linear negation to the set of all formulas as follows:

$$\begin{array}{lcllcl} (?A)^\perp & = & !(A^\perp) & (A \otimes B)^\perp & = & A^\perp \parr B^\perp \\ (!A)^\perp & = & ?(A^\perp) & (A \parr B)^\perp & = & A^\perp \otimes B^\perp \end{array}$$

The rules of the one-sided logic of multiplicative exponential linear logic are shown in Figure 5.3. Note the names of the rules; D stands for dereliction.

$$\frac{}{A, A^\perp}\,\mathsf{A} \qquad \frac{\Gamma, A \quad \Gamma', A^\perp}{\Gamma, \Gamma'}\,\mathsf{Cut}$$

$$\frac{\Gamma, A, B}{\Gamma, A \parr B}\,\mathsf{Par} \qquad \frac{\Gamma, A \quad \Gamma', B}{\Gamma, \Gamma', A \otimes B}\,\mathsf{Times}$$

$$\frac{\Gamma}{\Gamma, ?A}\,\mathsf{W} \qquad \frac{?\Gamma, A}{?\Gamma, !A}\,\mathsf{Box}$$

$$\frac{\Gamma, A}{\Gamma, ?A}\,\mathsf{D} \qquad \frac{\Gamma, ?A, ?A}{\Gamma, ?A}\,\mathsf{C}$$

$$\frac{\Gamma, A, B, \Gamma'}{\Gamma, B, A, \Gamma'}\,\mathsf{P}$$

Figure 5.3: Rules of one-sided MELL.

5.2.2 Definition of MELL Proof Nets

Proof nets [36] are designed as a graphic representation of one-sided linear logic sequent proofs. They are in a sense even superior to the sequent proofs, as insignificant variations in the order of application of logical rules do not show.

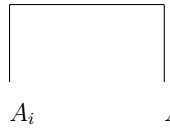
Now, we give the definition of the set of proof structures, of which the set of proof nets is a sub-class. Proof structures are introduced in [36], but this definition is inspired by [86]. We also introduce the way proof structures are usually drawn.

Definition 5.2.2 *A* proof-structure *P consists of the following objects:*

1. *Finitely many labelled occurrences of formulas (the labels are often omitted);*

2. *A certain number of links between these occurrences of formulas, which are of the following types:*

   Axiom link: $AX((A,i),(A^{\perp},j))$ *between an occurrence of* A *and an occurrence of* $A^{\perp}$*. This link has no premises and two conclusions, namely* (A,i) *and* $(A^{\perp},j)$*.*

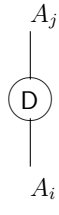
   Times link: $\otimes((A,i),(B,j),(A\otimes B,k))$*. The premises of the link are* (A,i) *and* (B,j) *and its conclusion is* $(A\otimes B,k)$*.*

   $$\frac{A_i \quad B_j}{(A\otimes B)_k}$$

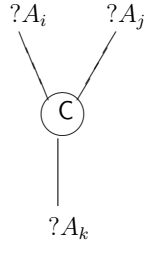
   Par link: $\parr((A,i),(B,j),(A\parr B,k))$*. The premises of the link are* (A,i) *and* (B,j) *and its conclusion is* $(A\parr B,k)$*.*

   $$\frac{A_i \quad B_j}{(A\parr B)_k}$$

   Dereliction link: $?((A,i),(?A,k))$*. The premise of this link is* (A,i) *and its conclusion is* $(?A,k)$*.*

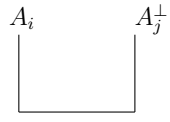
   Contraction link: $\mathsf{C}((?A,i),(?A,j),(?A,k))$*. The premises of this link are* $(?A,i)$ *and* $(?A,j)$ *and its conclusion is* $(?A,k)$*.*

Weakening link: $\mathsf{W}((?A, k))$. *This link has no premises and one conclusion, namely* $(?A, k)$.

Of course link: $!((A, i), !(A, k))$. *The premiss of this link is* (A, i)*, and its conclusion is* $(!A, k)$.

Cut link: $CUT((A, i), (A^{\perp}, j))$. *This link has the premises* (A, i) *and* $(A^{\perp}, j)$ *and no conclusions.*



Auxiliary link: $Aux((?A, i), (?A, k))$. *The premiss of this link is* $(?A, i)$*, and the conclusion is* $(?A, k)$.

We require that

- *every occurrence of a formula in the structure is the conclusion of exactly one link;*
- *every occurrence of a formula in the structure is the premise of at most one link. Formulas that are not the premise of a link, are called* terminal nodes.

3. *Boxes, which are sets of nodes of* P*. We require that:*
   - $P_{\restriction \mathcal{B}}$*, i.e.* P *restricted to* $\mathcal{B}$*, is a proof-structure.*
   - $P_{\restriction \mathcal{B}}$ *has exactly one terminal node that is the premiss of an of course link in* P*. This of course link is called* the principle port of $\mathcal{B}$.
   - *All other terminal nodes of* $P_{\restriction \mathcal{B}}$ *are premisses of auxiliary links in* P*. These auxiliary links are called* the auxiliary ports of $\mathcal{B}$.

- *There is no of course link that is not a principle port of some box.*
- *There is no auxiliary link that is not an auxiliary port of some box.*
- *Boxes are disjoint, or they are nested.*

We say that the nodes of $\mathcal{B}$ are inside box $\mathcal{B}$. *All other nodes of P are* outside box $\mathcal{B}$.

Definition 5.2.3 *The set of* proof nets *is an inductively defined subset of the set of proof-structures. The construction of proof nets is reflected schematically in Figure 5.4. Links to a Greek letter, like Γ, should be thought of as a collection of links to the formulas of Γ.*

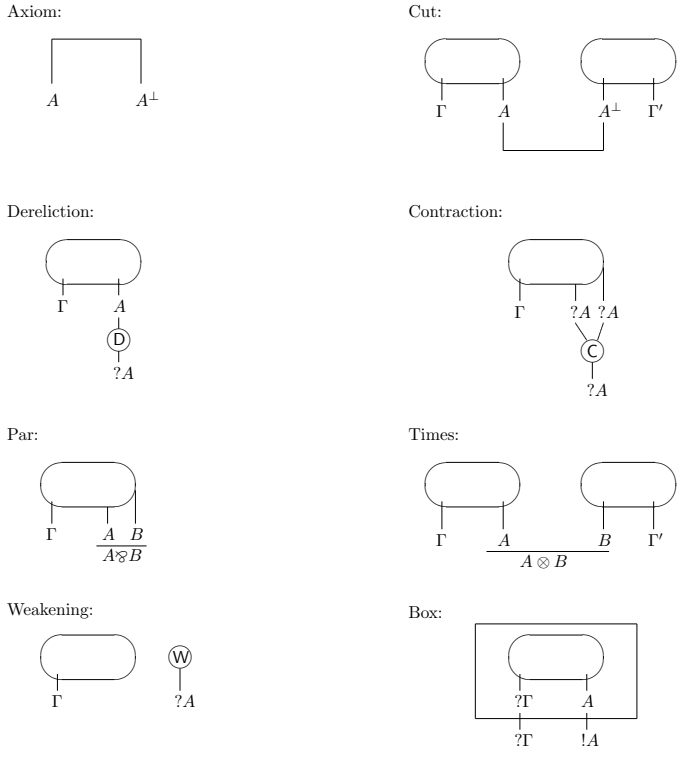


Figure 5.4: Proof nets

5.2.3 Reduction of Proof Nets

Cut-elimination of proof nets expresses the redundancy of the cut-rule. The transformations of proof nets are depicted in Figures 5.5 –5.10.

Remark the similarities with the transformations on Deduction Graphs: the c-b rule duplicates a box, just like the unsharing rule on deduction graphs;

the b-b rule absorbs one box into another, like the incorporation rule; the d-b rule opens a box, similar to what happens during the elimination of a safe cut in deduction graphs; the Ax-cut rule removes the repetition of a formula, like the repeat elimination rule. Note also that the w-b rule does not seem to correspond to any transformation rule of Deduction Graphs, as that formalism does not have garbage collection.

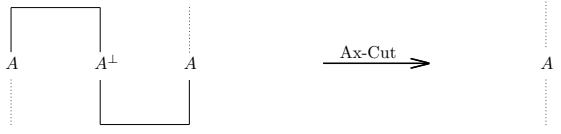


Figure 5.5: The Ax-cut rule

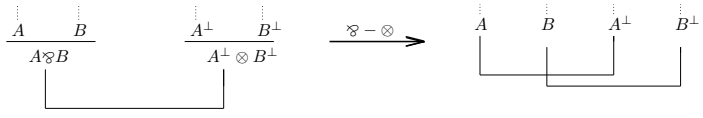


Figure 5.6: The $\parr - \otimes$ rule

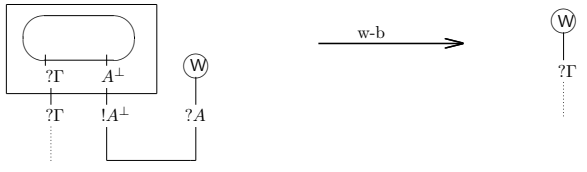


Figure 5.7: The w-b rule

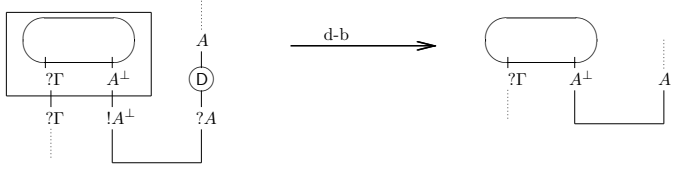


Figure 5.8: The d-b rule

Cut-elimination of proof nets is strongly normalising [36] [74].

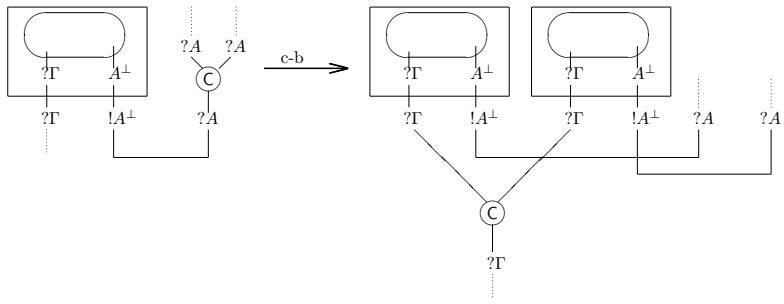


Figure 5.9: The c-b rule

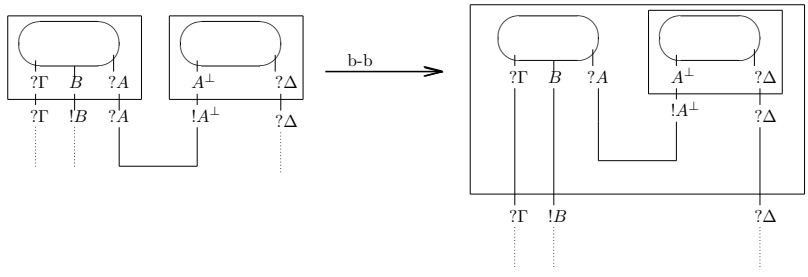


Figure 5.10: The b-b rule

5.3 Deduction Graphs with explicit sharing

There are some differences between DGs and proof nets:

1. DGs represent natural deduction derivations, whereas proof nets represent sequent calculus proofs.
2. Boxes of DGs border the scope of a local hypothesis, whereas proof nets guard the availability of resources.
3. A node of a DG may be shared any number of times, whereas the contraction links of proof nets contract only two nodes at one time.

We define a variant of DGs, called *deduction graphs with explicit sharing* (SGs) to close the third gab between deduction graphs and proof nets. If we have a *conclusion node* (i.e. a top-level node without incoming edges) (n, A), then we can add two nodes (m, A) and (k, A), which both have one outgoing edge to (n, A).

Furthermore, SGs allow edges that have as source and target one and the same node (**Loop** *edges*). This construction prevents the node from having incoming edges from other nodes, and thus serves to label garbage.

Definition 5.3.1 *The collection of* deduction graphs with explicit sharing *(*SG*) is the set of closed box directed graphs over* $\mathbb{N} \times \mathsf{Form}$ *inductively defined as follows.*

Axiom *A single node* (n, A) *is an* SG*,*

Join *If* G *and* G' *are disjoint* SG*s, then* $G'' := G \cup G'$ *is an* SG*.*

→-E *If* G *is an* SG *containing two conclusion nodes* $(n, A{\to}B)$ *and* (m, A)*, then the graph* $G' := G$ *with*

- *a new node* (p, B) *at the top level*
- *an edge* $(p, B) \longrightarrow\!\triangleright (n, A{\to}B)$*,*
- *an edge* $(p, B) \longrightarrow\!\triangleright (m, A)$*,*

is an SG*.*

Repeat *If* G *is an* SG *containing a conclusion node* (n, A)*, the graph* $G' := G$ *with*

- *a new node* (m, A) *at the top level,*
- *an edge* $(m, A) \longrightarrow\!\triangleright (n, A)$

is an SG*.*

Share *If* G *is an* SG *containing a conclusion node* (n, A)*, the graph* $G' := G$ *with*

- *two new nodes* (m, A)*,* (k, A) *at top level,*
- *an edge* $(m, A) \longrightarrow\!\triangleright (n, A)$*,*
- *an edge* $(k, A) \longrightarrow\!\triangleright (n, A)$

is an SG*.*

→-I *If* G *is an* SG *containing a conclusion node* (j, B) *and a free node* (m, A) *then the graph* $G' := G$ *with*

- *A box* $\mathcal{B}$ *with box-node* $(n, A{\to}B)$*, containing the node* (j, B) *as the only conclusion node, and* (m, A)*, and no other nodes that were free in* G*,*
- *An edge from the box node* $(n, A{\to}B)$ *to* (j, B)

is an SG *under the proviso that it is a well-formed closed box directed graph.*

Loop *If* G *is an* SG *containing the conclusions* (m, A) *and* (n, B) $(m \neq n)$*, then the graph* $G := G'$ *with an edge* $m \longrightarrow\!\triangleright m$ *is an* SG*.*

Furthermore, in the construction one should always take care that, if $k \longrightarrow\!\triangleright l$ *and* $k \in \mathcal{B}$ *and* $l \notin \mathcal{B}$*, then* k *has not been constructed in a* **Share***-step.*

So, apart from the four types of nodes A, I, E and R that we have already distinguished for deduction graphs in Chapter 2, we now also have S. A node is of a certain type, if it has been added in the corresponding construction rule of Definition 5.3.1. For example, a node is of type E if it has been added in a $\rightarrow$-E-step.

Lemma 5.3.2 *G is an* SG *if and only if the following hold*

1. *G is a finite closed box directed graph,*
2. *every node has at most two incoming edges. If it has two incoming edges, they both come from nodes with the same label, that have only one outgoing edge;*
3. *there is a box-topological ordering $>$ of G, such that, for all m, n, k if $m \longrightarrow\!\triangleright\, k$ and $n \longrightarrow\!\triangleright\, k$ then n is an immediate successor of m or m is an immediate successor of n.*
4. *every node n of G is of one of the following five types:*

   A *n has no outgoing edges.*

   E *n has label B and has exactly two outgoing edges: one to a node $(m, A{\rightarrow}B)$ and one to a node (p, A), both within the scope of n.*

   R *n has label A and has exactly one outgoing edge, which is to a node (m, A) that is within the scope of n and has exactly one incoming edge. The outgoing edge traverses at most one border of a box.*

   S *n has label A and has exactly one outgoing edge, which is to a node (m, A) that is within the scope of n and has exactly two incoming edges. The outgoing edge does not traverse any border of a box.*

   I *n is a box-node of a box $\mathcal{B}$ with label $A{\rightarrow}B$ and has exactly one outgoing edge, which is to a node (j, B) inside the box $\mathcal{B}$ with no other ingoing edges. There is one node inside the box without outgoing edges labeled A and there are no other nodes without outgoing edges inside the box.*

Proof Similar to the proof of Lemma 2.2.7

Definition 5.3.3 *A* conclusion node*, is a top level node without incoming edges.*

Definition 5.3.4 *A top-level node (n, A) such that $n \longrightarrow\!\triangleright\, n$ is called a* fake conclusion node.

It is easy to turn a deduction graph into a deduction graph with explicit sharing, simply by making the implicit sharing in DGs explicit via S-nodes. This is indicated in Figure 5.11. That the changes in sharing, looping and arrow introduction are not serious, is stated more precisely in the following lemma.

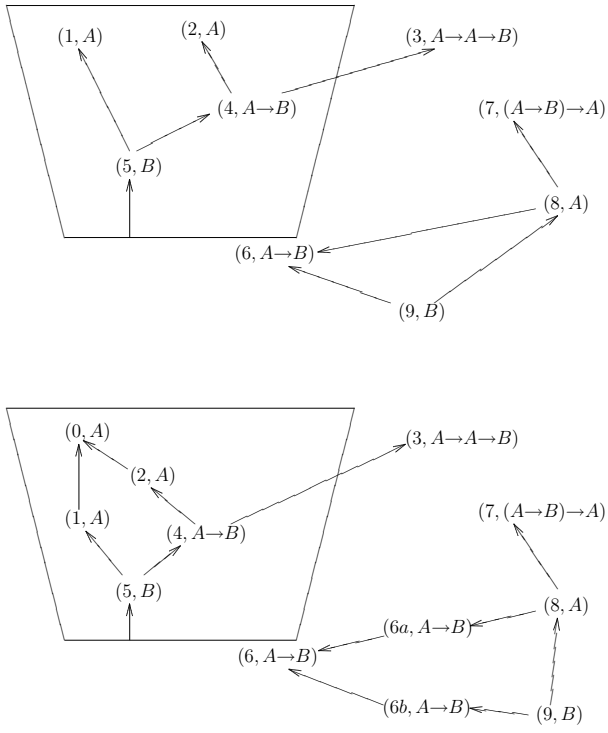


Figure 5.11: Example of a deduction graph (above) and the same graph with explicit sharing constructions (beneath)

Lemma 5.3.5 *If G is an* DG *with free nodes Γ and conclusion nodes Δ, then there exists an* SG *G' with free nodes Γ and conclusion nodes Δ.*

However, because of the change in the arrow introduction together with the new loop-rule, SGs have the following property:

Lemma 5.3.6 *If G, G' are* SG*s, Γ are the conclusion nodes of G and $G \Rightarrow G'$, then Γ are the conclusion nodes of G'.*

We now modify the process of cut-elimination, because the deduction graphs with explicit sharing are not closed under the cut-elimination rules we have defined for DGs. For example, we start the process by eliminating all repeats that separate the I-node of the cut from the E-node. Because in the deduction graphs with explicit sharing it is required that outgoing edges from S-nodes do not cross the border of any box, we now have to postpone the removal of some R-nodes. Therefore the process will transform the SG levelwise. The changes only affect the order in which we make the cut explicit. The elimination of the

safe cut itself remains the same. A *safe cut* is the situation where we have an E-node where the edge to the →-formula points to an I-node at the same level, so we have an →-introduction followed immediately by an →-elimination. In that case, the cut can be removed by a simple reordering of edges and the removal of some nodes. This is discussed in detail for DGs in Chapter 3; the situation for SGs is slightly simpler.

Definition 5.3.7 *A cut in an* SG G *is a subgraph of* G *consisting of:*

- *A box-node* $(n, A{\to}B)$,
- *A node* (p, B),
- *A node* (m, A),
- *A sequence of* R *and* S *nodes* $(s_0, A{\to}B), \ldots, (s_i, A{\to}B)$,
- *Edges* $(p, B) \longrightarrow\!\triangleright (s_i, A{\to}B) \longrightarrow\!\triangleright \ldots \longrightarrow\!\triangleright (s_0, A{\to}B) \longrightarrow\!\triangleright (n, A{\to}B)$,
- *An edge* $(p, B) \longrightarrow\!\triangleright (m, A)$.

The sequence $(p, B) \longrightarrow\!\triangleright (s_i, A{\to}B) \longrightarrow\!\triangleright \ldots \longrightarrow\!\triangleright (n, A{\to}B)$ *is called the* cut-sequence.

Definition 5.3.8 (Cut hidden by repeats) *Let* G *be an* SG *and let* c *be a cut of* G *and let* S *be that part of the cut-sequence that is at the same level as* $(n, A{\to}B)$. *If* s_j *is an* R*-node in* S, *for some* $0 \leq j \leq i$, *then the repeat-elimination at* s_j *is obtained by:*

- *When an edge points to* s_j, *redirect it to* s_{j-1} *(or to* n, *if* $j = 0$*);*
- *Remove* s_j.

Definition 5.3.9 (Cut hidden by sharing) *Let* G *be an* SG *with a cut* c *that contains a box* $\mathcal{B}$ *with box-node* $(n, A{\to}C)$ *and in-going edges from* p_1, p_2. *Then the* unsharing of G at nodes n, p_1, p_2 *is obtained by:*

- *removing* $\mathcal{B}$,
- *adding copies* $\mathcal{B}'$ *and* $\mathcal{B}''$ *of* $\mathcal{B}$ *including copies of the nodes reachable within one step,*
- *Connect the copies outside the boxes, to the original nodes. (thus if we had* $q \longrightarrow\!\triangleright m$ *with* $q \in \mathcal{B}$, $m \notin \mathcal{B}$ *and* m' *is the copy of* m *connected to* $\mathcal{B}'$, *and* m'' *is the copy of* m *connected to* $\mathcal{B}''$, *then we add the edges* $m' \longrightarrow\!\triangleright m$ *and* $m'' \longrightarrow\!\triangleright m$*),*
- *replacing* n'' *by* p_1 *and replacing* n' *by* p_2.

Figure 5.12 shows the unsharing of the SG in Figure 5.11.

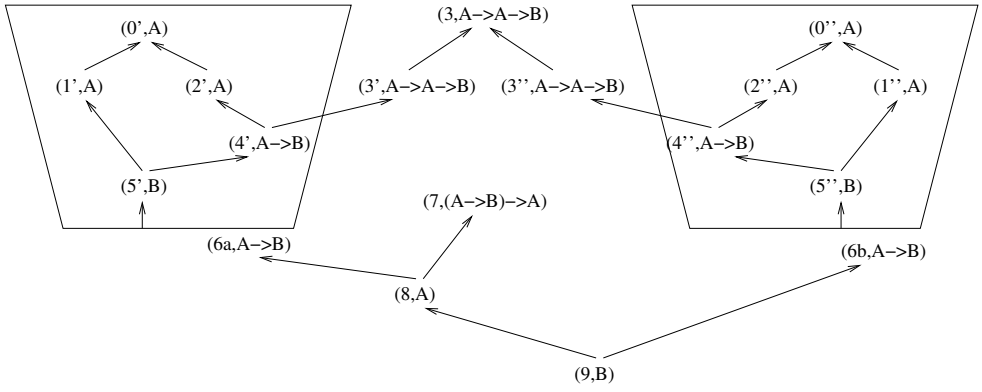


Figure 5.12: Unsharing of the SG in Figure 5.11

Definition 5.3.10 (Cut hidden by a depth conflict; incorporation) *We have a* depth conflict *in the* SG G *with cut* c*, if* $(n, A{\to}B)$ *has one incoming edge, which comes from a node at another level. Let* $\mathcal{B}$ *the box of which* $(n, A{\to}B)$ *is the box-node. In that case the* incorporation of G at c *is obtained by putting* $\mathcal{B}$ *at one level deeper.*

Figure 5.13 shows an SG with a cut hidden by a depth conflict; Figure 5.14 shows its incorporation at $7, 12$.

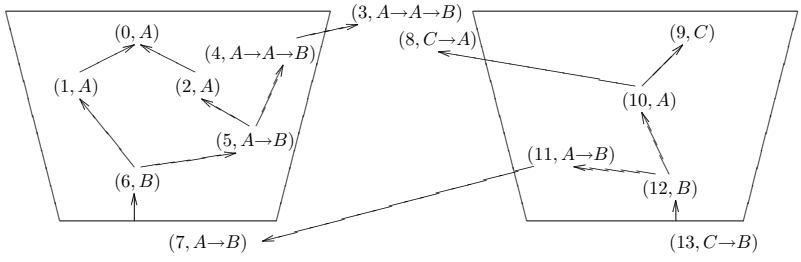


Figure 5.13: SG with cut hidden by a depth conflict

Definition 5.3.11 *Given an* SG G *with a cut* c*,* the process of eliminating the cut c *is the following:*

1. *(Repeat-elimination) As often as possible, perform the repeat-elimination step as described in Definition 5.3.8.*
2. *(Unsharing) As often as possible, perform the unsharing step as described in Definition 5.3.9.*
3. *(Incorporation) If possible, perform the incorporation step as described in Definition 5.3.10.*

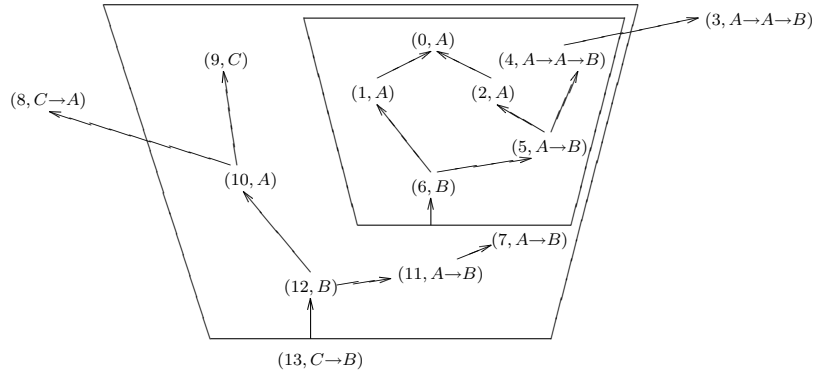


Figure 5.14: Incorporation of the SG in Figure 5.13

4. *(Moving up one level) If c is not yet safe, repeat the procedure, starting at 1.*

5. *(Eliminating the safe cut) Eliminate the safe cut.*

The proof of strong normalisation of the process of cut-elimination on DGs in Chapter 4 makes only use of the fact that it always ends in the elimination of a safe cut; the order in which we apply the other steps –repeat-elimination, unsharing and incorporation– plays no role. As the process of cut-elimination on SGs still ends in the elimination of a safe cut, the proof goes through without many adjustments.

5.4 Translation of Formulas

To translate SGs to proof nets, we first have to translate the formulas of minimal propositional logic into linear logic formulas. This is done via the translation $(-)^*$ defined as follows.

$$\begin{aligned} A^* &:= !A \text{ for } A \text{ an atom} \\ (A{\rightarrow}B)^* &:= !(A^{*\perp} \parr B^*) \end{aligned}$$

Because we want to unify deduction graph boxes with proof net boxes, other translations seem less suitable.

5.5 Direct translation to Proof Nets

This section presents the first translation from SGs to proof nets, which is direct. In the definition of the translation we will use the *rank* of a node: The conclusion nodes of an SG we give rank 0, and – roughly speaking – a node is given rank

$i+1$ if there is an arrow to it from a node with rank i. We have to be careful in the case of sharing.

Definition 5.5.1 (Closure) *Let G be an* SG *and let $\mathcal{B}$ be a box of G. The* closure of $\mathcal{B}$, *called $\overline{\mathcal{B}}$, is the graph consisting of all nodes inside $\mathcal{B}$ plus all nodes that can be reached from within $\mathcal{B}$ in one step.*

$\overline{\mathcal{B}}$ is also an SG. (This follows from results in Chapter 2.)

Definition 5.5.2 *Given a deduction graph G, the rank of the nodes in G is defined as follows.*

- *If n is a conclusion node,* $\mathsf{rank}(n) := 0$.
- *If n is a fake conclusion node,* $\mathsf{rank}(n) := 0$.
- *If n is not an* S*-node or an* I*-node and $n \rightarrowtriangle m$, then* $\mathsf{rank}(m) := \mathsf{rank}(n)+1$.
- *If n is a* S*-node and $n \longrightarrow\!\!\triangleright\, m$ and $k \longrightarrow\!\!\triangleright\, m$, then*

$$\mathsf{rank}(m) := \mathsf{max}(\mathsf{rank}(n), \mathsf{rank}(k)) + 1.$$

- *If n is a* I*-node with box $\mathcal{B}$, then* $\mathsf{rank}(j) := \mathsf{rank}(n)+1$ *for all nodes $j \in \overline{\mathcal{B}}$.*

The rank of an SG *G,* $\mathsf{rank}(G)$, *is the maximum of the rank of its nodes.*

Lemma 5.5.3 *Let G be an* SG *with* $\mathsf{rank}(G) = i+1$ *for some i.*

- *Suppose (n, B) of rank i is an* E*-node with edges to (m, A) and $(l, A \rightarrow B)$. Then $G \setminus m, l$ is an* SG.
- *Suppose $(n, A \rightarrow B)$ of rank i is an* I*-node of box $\mathcal{B}$. Then both $\overline{\mathcal{B}}$ and $G \setminus \overline{\mathcal{B}}$ are* SG*s.*
- *Suppose (m, A) of rank i is an* S*-node with an edge to (n, A). Then $G \setminus n$ is an* SG.
- *Suppose (n, A) of rank i is an* R*-node with an edge to (m, A). Then $G \setminus m$ is an* SG.

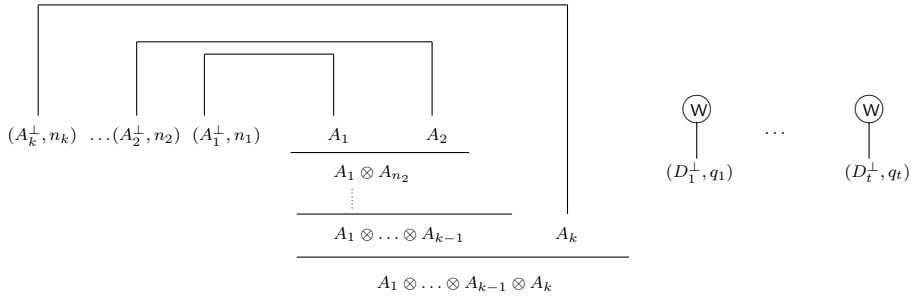
Proof By Lemma 5.3.2.

Definition 5.5.4 *Given an* SG *G we define a proof net (with labelled nodes) $\mathcal{V}(G)$, which gives the proof net associated to G, by induction on the number of nodes of G.*
The invariant that we maintain is the following. If G has
– free nodes $(n_1, A_1), \ldots (n_k, A_k)$,
– conclusion nodes $(m_1, B_1), \ldots, (m_l, B_l)$,
then $\mathcal{V}(G)$ has
– terminal nodes $B_1^* \otimes \ldots \otimes B_l^*$, and $A_1^{*\perp}, \ldots A_k^{*\perp}$, labelled $n_1, \ldots, n_k$ respectively.

To relieve the notational burden, we just write A instead of A^* in the proof nets. The definition of $\mathcal{V}(G)$ is as follows. We make a case-distinction according to $\mathsf{rank}(G)$.

Case $\mathsf{rank}(G) = 0$**.**

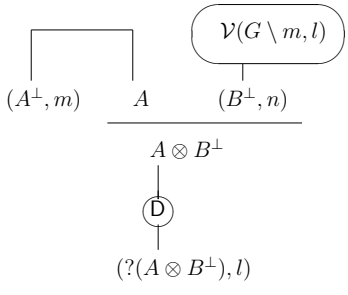
We only have to consider the conclusions $(n_1, A_1), \ldots (n_k, A_k)$ of G and the fake conclusions $(q_1, D_1), \ldots, (q_t, D_t)$. Now $\mathcal{V}(G)$ is the proof net containing axiom links $A_s - A_s^\perp$ (with $A_s^\perp$ labelled by n_s for $1 \leq s \leq k$) and the $A_1, \ldots, A_n$ nodes made into a tensor product $A_1 \otimes \ldots \otimes A_n$ by $n-1$ tensor links. Furthermore it contains weakening links on nodes $D_s^\perp$ (labelled n_s) for $1 \leq s \leq t$.



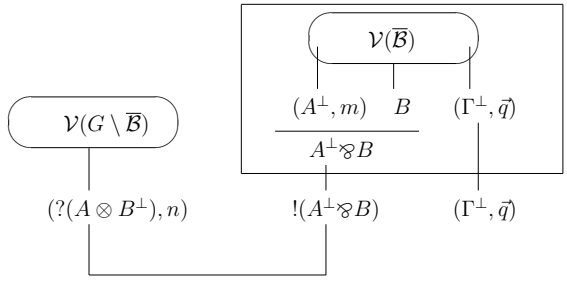
Case $\mathsf{rank}(G) = i + 1$ **for some** i**.**

Suppose that $\mathcal{V}(F)$ has already been defined for SG*s F with less nodes than G. We consider the non-*A*-nodes of rank i in some order (say the box-topological ordering). We distinguish cases according to the type of the node.*

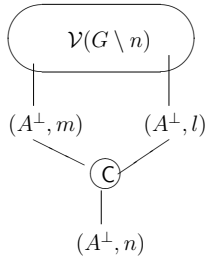
- E. *Suppose node (n, B) is of rank i with two edges to (m, A) and $(l, A{\to}B)$. So the nodes m and l are of rank $i+1$. Then $\mathcal{V}(G)$ is defined as follows:*



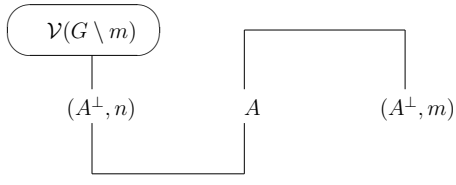
- I. *Suppose node $(n, A{\to}B)$ of rank i is a box node of box $\mathcal{B}$ with an edge to (j, B) and let (m, A) be the discharged node. Let $(\vec{q}, \Gamma)$ be the nodes that can be reached from within $\mathcal{B}$ with one edge. Then $\mathcal{V}(G)$ is defined as follows:*



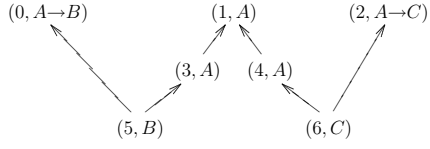
- S *Suppose node* (m, A) *of rank* i *is an* S*-node with an edge to* (n, A)*. Let* (l, A) *be the other* S*-node with the same target. Then* $\mathcal{V}(G)$ *is defined as follows:*

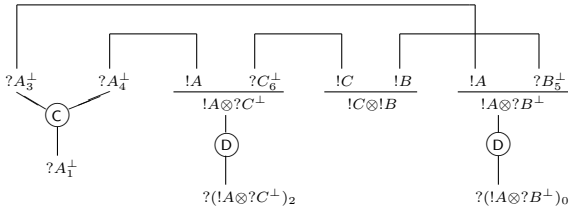


- R *Suppose* (n, A) *of rank* i *is an* R*-node with an edge to* (m, A)*. Then* $\mathcal{V}(G)$ *is defined as follows:*



Example 5.5.5 *Here we see an example of a simple* SG G *and the translation* $\mathcal{V}(G)$*, written out completely.*



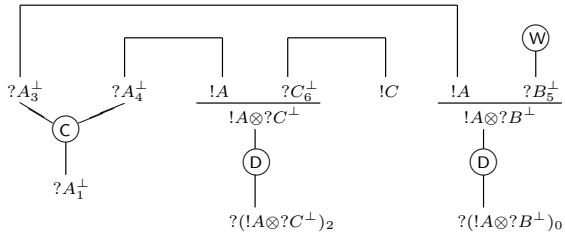


5.6 Context Nets and Closed One-Liners

In Chapter 4 we have made a translation that gives the simply typed λ-term associated to *a node* in a deduction graph. This solved the discrepancy that deduction graphs can have many conclusions, whereas simply typed λ-terms have just one type. Another solution could have been to make a translation from deduction graphs to typed λ-terms with conjunction types. So then we would associate a λ-term of type $B_0 \wedge B_1 \wedge \ldots \wedge B_k$ to a deduction graph with conclusions $(n_0, B_0), (n_1, B_1), \ldots, (n_k, B_k)$. In Chapter 4 we have not investigated this, but instead we have made a translation from deduction graphs to *contexts*, solving the problem of the many conclusions in a different way. Contexts do not have types (they can only be "well-formed"), but they can obtain a type later, by filling in a variable associated to a node of a deduction graph.

The direct translation of Section 5.5, can be seen as the proof net equivalent of the translation from DGs to λ-terms with conjunction types. We can also define a translation to proof nets from SGs in a specific conclusion, as shown in Example 5.6.1. This can be seen as the proof net equivalent of the translation to simply typed terms.

Example 5.6.1 *Here we see a translation of the* SG *in Example 5.5.5 in node 6. The translation of* $(5, B)$ *is weakened, and thus it plays the role of a fake conclusion.*



We do not go into this or other translations to proof nets, but we explore a translation to the net equivalent of contexts: context nets. Not only are context nets closer to deduction graphs as they have both "conclusions" and "assumptions", but this translation also generalises the translations to proof nets, because we can recover them by "gluing" something to the context net; the net equivalent of filling a hole (see Section 5.8).

We define *context structures*, an extension of Girard's proof structures [36]. It allows *initial nodes*, i.e. nodes that are not a conclusion. Similar extensions can be found in Danos and Regnier [24] and Puite [72]. Intuitively, this restores the duality found in two-sided sequents, as initial nodes of a context net can be seen as the formula occurrences left of the turn-style and the terminal nodes can be seen as the occurrences on the right of it. Only when initial nodes are inside a box, the correspondence to two-sided logic is not clear.

Another way to look at it, is to consider a context structure as an "open proof structure": If we connect proof structures with corresponding terminal nodes to the initial nodes of the context structure, we get a proof structure. In order to obtain structures that are subgraphs of proof structures, and in contrast to Puite, we do not expand the set of links. Another difference is that we consider an extension of proof nets of MELL (instead of MLL).

Definition 5.6.2 (Context Structures) *A* context structure *for MELL is a proof structure for MELL, with the difference that nodes that are not a conclusion, are allowed. These nodes are called* initial nodes*.*

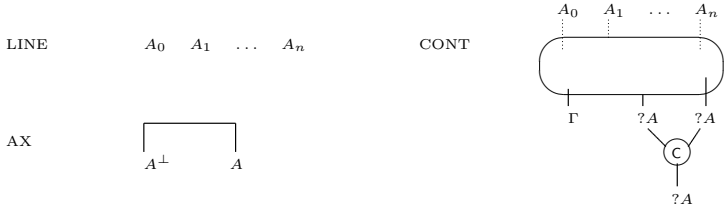
Just as the construction rules of proof nets determine the subset of proof structures that are correct, which means that they agree with a sequent calculus proof, we would like to have construction rules for "context nets", which somehow should determine correct context structures. One could say that context structures are correct, when they agree with a two-sided sequent calculus proof. Another notion of correctness is to say that, if a context net happens to be a proof structure, it is a proof net. It is this latter notion we adopt and we will give the construction rules, which are essentially the rules put forward by David and Kesner [25].

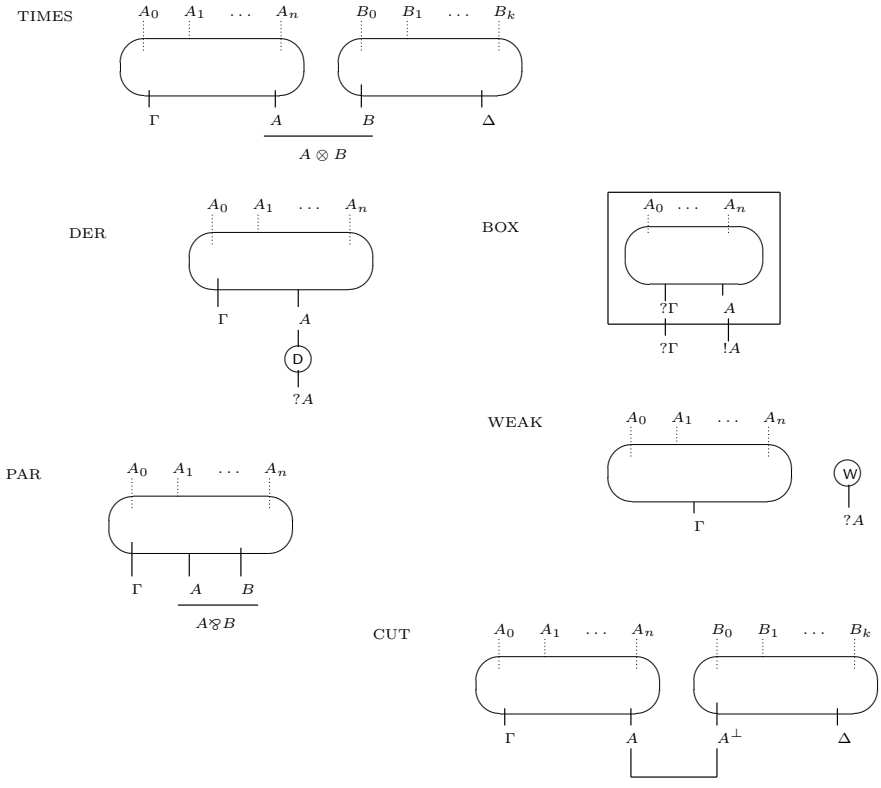
The rules of context nets are at the same time a bit rigid, as they consider only the terminal nodes, leaving the initial nodes untouched throughout the construction, and extremely useful, as the class of context structures singled out by these construction rules can easily be seen to be correct.

To understand the LINE-rule in the following definition as a two-sided sequent rule $\Gamma \vdash \Gamma$, we have to enrich the sequent calculus with the mix-rule [24]:

$$\frac{\Delta \vdash \Sigma \quad \Delta' \vdash \Sigma'}{\Delta, \Delta' \vdash \Sigma, \Sigma'}\,\mathsf{Mix}$$

Definition 5.6.3 (Context Nets) Context nets *are inductively defined as follows.*





Lemma 5.6.4 (Correctness of Context Nets) *Every context net without initial nodes is a proof net.*

Proof Immediate from the definition of context nets.

We now focus our attention on a subclass of context nets, the *closed one-liners*. A closed one-liner can be seen as a context net Θ such that, if we connect every initial node of it to a corresponding terminal node of *one* proof net, say Σ, we get again a proof net.

Definition 5.6.5 (Closed one-liners) *A* closed one-liner *is a context net that has no initial nodes inside boxes and that can be formed by using the LINE-rule at most once.*

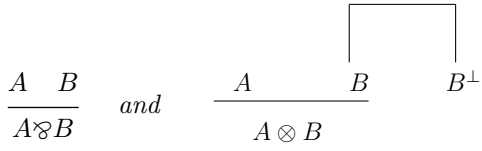
Example 5.6.6

- *The context structure*

$$\frac{A \quad B}{A \otimes B}$$

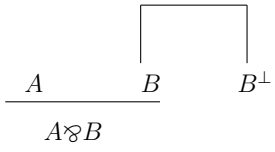
is not *a closed one-liner, because the initial nodes A and B originate from* different *applications of the LINE-rule: They are thought of as terminal nodes of* different *proof nets.*

- *The context structures*



are closed one-liners. In the first structure the initial nodes A and B originate from one and the same application of the LINE-rule: They are thought of as terminal nodes of one *proof net. The second context net has only one initial node, which makes the case trivial.*

- *The context structure*

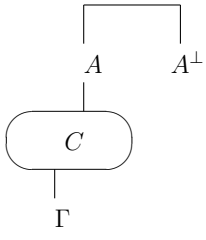


is not *a closed one-liner. Note that it is not even a context net.*

As remarked before, the rules of context nets, and hence the rules of closed one-liners, are a bit rigid, because they leave the initial nodes untouched throughout the construction. The following lemma gives some closure properties that enable a more flexible handling of closed one-liners.

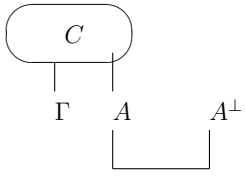
Lemma 5.6.7 (Closure properties)

ax *Suppose C is a closed one-liner with initial node A, and terminal nodes Γ. Then*



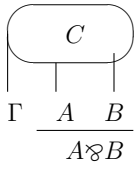
is a closed one-liner too.

cut *Suppose C is a closed one-liner with terminal nodes Γ, A (and no initial nodes). Then*



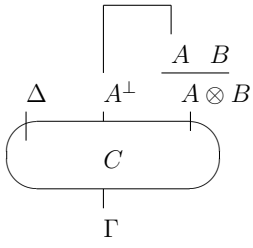
is a closed one-liner too.

par *Suppose C is a closed one-liner with terminal nodes Γ, A, B (and no initial nodes). Then*



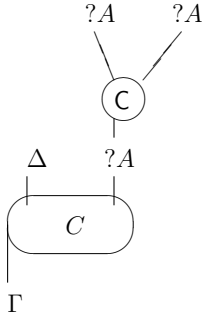
is a closed one-liner too.

times *Suppose C is a closed one-liner with initial nodes $\Delta, A^{\perp}, A \otimes B$ and terminal nodes Γ. Then*



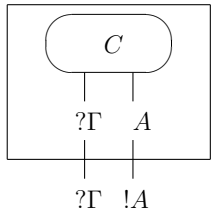
is a closed one-liner too.

cont *Suppose C is a closed on-liner with initial nodes $\Delta, ?A$ and terminal nodes Γ. Then*



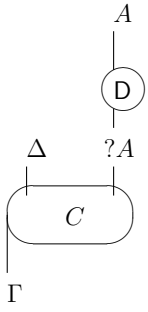
is a closed one-liner too.

box *Suppose C is a closed one-liner with terminal nodes $?\Gamma, A$ (and no initial nodes). Then*



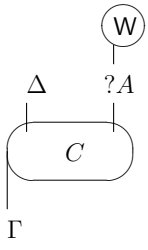
is a closed one-liner too.

der *Suppose C is a closed one-liner with initial nodes $\Delta, ?A$ and terminal nodes Γ. Then*



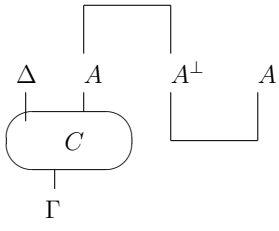
is a closed one-liner too.

weak *Suppose C is a closed one-liner with initial nodes $\Delta \neq \emptyset, ?A$ and terminal nodes Γ. Then*



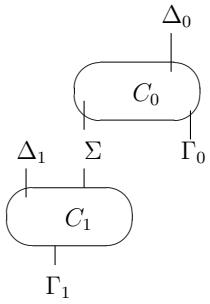
is a closed one-liner too.

rep *Suppose C is a closed one-liner with initial nodes Δ, A and terminal nodes Γ. Then*



is a closed one-liner too.

comb *Suppose C is a closed one-liner with initial nodes Δ_0 and terminal nodes Σ, Γ_0, and C_1 is a closed one-liner with initial nodes Δ_1, Σ, and terminal nodes Γ_1, such that $\Delta_1 \cup \Sigma \neq \emptyset$ and if $\Delta_0 = \emptyset$, then $\Delta_1 = \emptyset$. Then*



is a closed one-liner too.

Proof The idea is to change the formulas that are involved in the application of the LINE-rule, if it exists. To make this work, it is sometimes important that there exists indeed an application of the LINE-rule in C, or that there is still an application of the LINE-rule in the resulting net: Hence the side-conditions for the closure properties *weak* and *comb*.

ax The new closed one-liner can be obtained by replacing the application of the LINE-rule in the construction of C by an AX-rule.

cut C has no initial nodes. So a construction of C does not make use of the LINE-rule. We get the new closed one-liner by applying LINE on $A^\perp$ and doing a CUT with this on C.

par Immediate using PAR.

times The new closed one-liner can be obtained by replacing the application of the LINE-rule in the construction of C, by an application of the LINE-rule on Δ, B, an AX-rule and an TIMES-rule.

cont Replace the application of the LINE-rule on $\Delta, ?A$ by an application of the LINE-rule on $\Delta, ?A, ?A$, followed by an application of the CONT-rule.

box Immediate using BOX.

der Replace the application of the LINE-rule on $\Delta, ?A$ by an application of the LINE-rule on Δ, A, followed by an application of the DER-rule.

weak Replace the application of the LINE-rule on $\Delta, ?A$, by an application of the LINE-rule on Δ, followed by an application of the WEAK-rule on $?A$. Note that the WEAK-rule can only be applied, because $\Delta \neq \emptyset$.

rep Let the application of the LINE-rule of the construction of C be followed by an application of the CUT-rule on A and an $A^\perp$, the latter having been created by an application of the AX-rule.

comb If $\Delta_1 \neq \emptyset$, replace the application of the LINE-rule of the construction of C_0 (which must exist!) by an application of the LINE-rule on Δ_0, Δ_1. Then we get a closed one-liner, say C_0' with terminal nodes $\Gamma_0, \Sigma, \Delta_1$.

Replace subsequently the application of the LINE-rule on Δ_1, Σ in the construction of C_1, by C_0'. If $\Delta_0 = \Delta_1 = \emptyset$ it suffices to replace the application of the LINE-rule in the construction of C_1, which exists because $\Sigma \neq \emptyset$, by the construction of C_0.

5.7 Translation via context nets

5.7.1 From Deduction Graphs to Context Nets

Definition 5.7.1 *Given a deduction graph G, we define a context net $\mathcal{I}(G)$ associated to G by induction on the construction of G. The invariant we maintain is:*

> If G is a deduction graph with assumptions Δ and conclusions Γ, then $\mathcal{I}(G)$ is a closed one-liner with terminal nodes $\Delta^{*\perp}$ and initial nodes $\Gamma^{*\perp}$.

We put a picture of G on the left and a picture of $\mathcal{I}(G)$ on the right. Again, A stands for A^* and B stands for B^*.

Axiom *The last step is an* **Axiom** *step, then applying the LINE-rule on $A^\perp$:*

(n, A) $(A^\perp, n)$

Join *The last step is a* **Join** *step:*

The invariant is maintained because of the closure property join *with $\Sigma = \emptyset$. Because G' and G'' both have conclusions, $\mathcal{I}(G')$ and $\mathcal{I}(G'')$ both have initial nodes.*

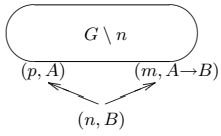
Repeat *The last step is a* **Repeat** *step:*

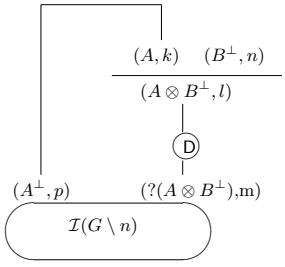
The invariant is maintained because of the closure property rep.

Share *The last step is a* **Share** *step:*

The invariant is maintained because of the closure property cont.

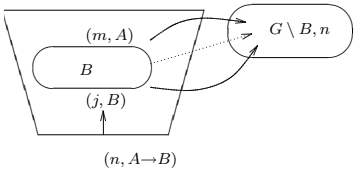
→-E *The last step is a* **→-E** *step:*

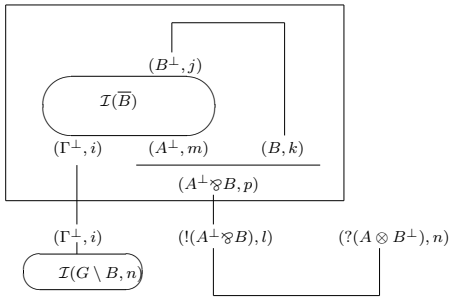




The invariant is maintained because of the closure properties der *and* times.

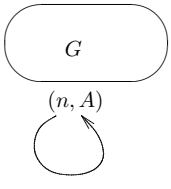
→-I *The last step is a* **→-I** *step:*

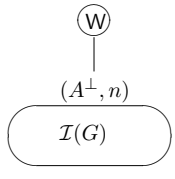




The invariant is maintained because of the closure properties ax, par, box, cut *and* comb.

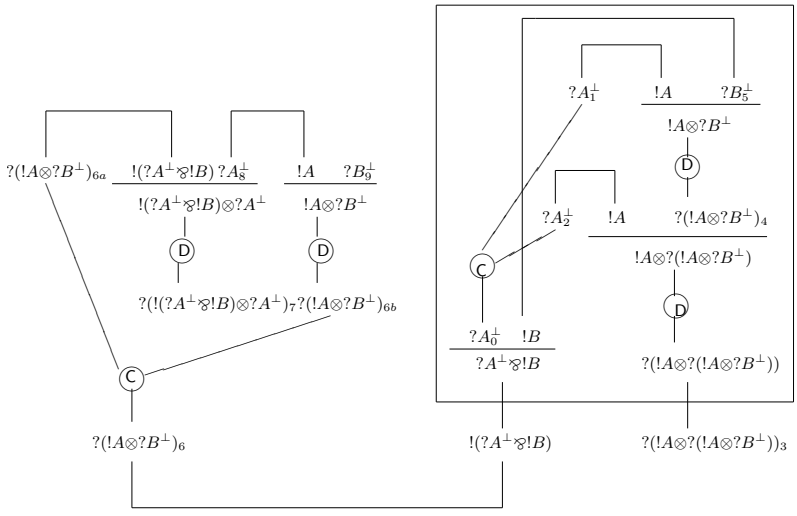
Loop *The last step is a* **Loop** *step:*





The invariant is maintained because of the closure property weak.

Example 5.7.2 *We show the proof net that we obtain via* $\mathcal{I}(-)$ *from the example in Figure 5.11.*



5.7.2 Discussion

Unlike the direct translation $\mathcal{V}$, the translation via context nets $\mathcal{I}$ treats assumptions and conclusions in a uniform way: Assumptions Γ of the deduction graph (SG) associate to terminal nodes $\Gamma^{*\perp}$ in the context net and conclusions Δ associate to initial nodes $\Delta^{*\perp}$.

This immediately triggers two questions:

1. Why do formulas that appear in the deduction graph correspond to the *negation* of their translations, and not to their translations themselves?

2. Why does the (construction of a) deduction graph correspond to (a construction of a) deduction net *in an up-side down way*, i.e. conclusions (at the bottom of the graph) correspond to initial nodes (at the top of the net) and free nodes (at the top of the graph) correspond to terminal nodes (at the bottom of the net)?

Both these disagreements stem from choices: Neither of them is essential.

The answer to the first lies in the origin of proof nets. Proof nets are a graphical representation of *right*-sided linear logic sequent calculus (Section 5.2). If proof nets would have been defined starting from *left*-sided linear logic sequent calculus, then formulas in deduction graphs would correspond to their true translation in the deduction net. Figure 5.15 presents the rules of left-sided linear logic.

The second question is easily solved by drawing deduction nets (or deduction graphs) up-side down. I do not believe that this is a way to mask "semantic" differences (assumption/ conclusion) by syntactical means (up/ down). Rather I think that drawing the nets or the graphs up-side down makes the deep conformity between deduction graphs and (left-sided) context nets even clearer.

$$\frac{}{A, A^{\perp}}\,\mathsf{A} \qquad \frac{\Gamma, A \quad \Gamma', A^{\perp}}{\Gamma, \Gamma'}\,\mathsf{Cut}$$

$$\frac{\Gamma, A, B}{\Gamma, A \otimes B}\,\mathsf{Times} \qquad \frac{\Gamma, A \quad \Gamma', B}{\Gamma, \Gamma', A \parr B}\,\mathsf{Par}$$

$$\frac{\Gamma}{\Gamma, !A}\,\mathsf{W} \qquad \frac{!\Gamma, A}{!\Gamma, ?A}\,\mathsf{Box}$$

$$\frac{\Gamma, A}{\Gamma, !A}\,\mathsf{D} \qquad \frac{\Gamma, !A, !A}{\Gamma, !A}\,\mathsf{C}$$

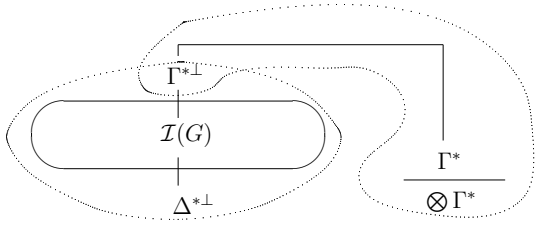
$$\frac{\Gamma, A, B, \Gamma'}{\Gamma, B, A, \Gamma'}\,\mathsf{P}$$

Figure 5.15: Rules of *left*-sided linear logic.

5.8 Connecting the two Translations

The translations $\mathcal{V}(G)$ and $\mathcal{I}(G)$ are different in their construction. If G is a deduction graph with assumptions Δ and conclusions Γ, then $\mathcal{I}(G)$ is a deduction net with terminal nodes $\Delta^{*\perp}$ and initial nodes $\Gamma^{*\perp}$. By adding axiom links and tensor nodes, we turn this into a proof net with terminal nodes $\Delta^{*\perp}, \bigotimes \Gamma^{*}$, where $\bigotimes \Gamma^{*}$ is the formula obtained by putting a tensor between all formulas in Γ^{*}.

Definition 5.8.1 *Let G be an* SG*. We define $\mathcal{J}(G)$ to be the proof structure that is schematically presented in the following figure:*



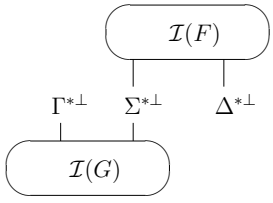
Lemma 5.8.2 *For all* SG*s G, $\mathcal{J}(G)$ is a proof net.*

Proof The two dashed parts are closed one-liners. By the closure property *comb*, $\mathcal{J}(G)$ is a closed one-liner too. Because $\mathcal{J}(G)$ does not have initial nodes, it is a proof net (Lemma 5.6.4).

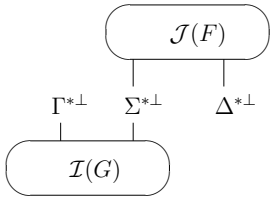
We will see that $\mathcal{J}$ and $\mathcal{V}$ give the same results.

Lemma 5.8.3 (Gluing) *Let G be an* SG *with conclusions Σ, Γ and let F be an* SG *with assumptions Σ, Δ. Define $H := G \cup F$. Then*

1. $\mathcal{I}(H) =$



2. $\mathcal{J}(H) =$

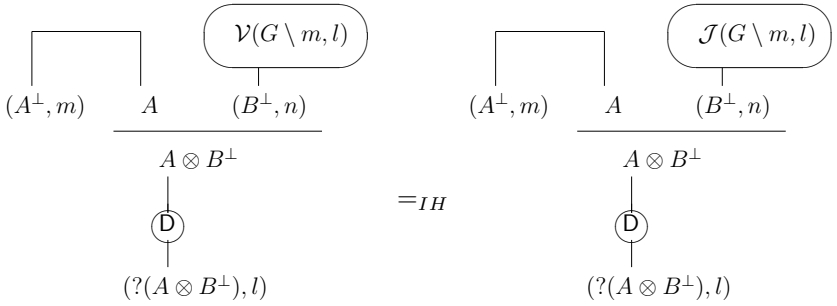


Proof By induction on the number of non A-nodes of F.

Theorem 5.8.4 *Let G be an* SG*. Then $\mathcal{V}(G) = \mathcal{J}(G)$.*

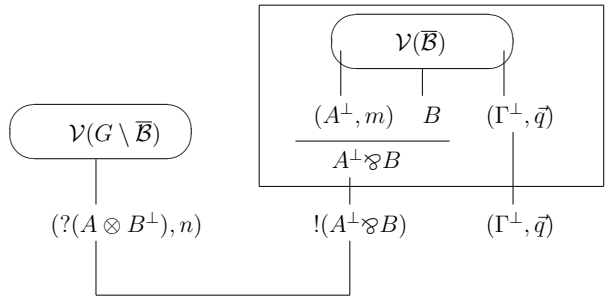
Proof The proof is by induction on the number of nodes of G. We make a case-distinction to $\mathsf{rank}(G)$.
Case $\mathsf{rank}(G) = 0$.
Then G has been constructed using **Axiom**, **Join**, and **Loop**. We easily verify that $\mathcal{V}(G) = \mathcal{J}(G)$.
Case $\mathsf{rank}(G) = i + 1$ for some i.
Suppose that $\mathcal{V}(F)$ has already been defined for SGs F with less nodes than G. We consider the non-A-nodes in some order (say the box-topological ordering). We distinguish cases according to the type of the node. We treat here the E-case and the I-case.

- E Suppose node (n, B) is of rank i with an edge to (m, A) and an edge to $(l, A \to B)$. Let F be the graph consisting of the nodes n, m, and l and the edges $n \longrightarrow\!\triangleright\, m$ and $n \longrightarrow\!\triangleright\, l$. Then $\mathcal{V}(G) =$



Note that this is $\mathcal{J}(G \setminus m, l)$ glued to $\mathcal{I}(F)$ and by Lemma 5.8.3 we are done.

- I Suppose node $(n, A \rightarrow B)$ of rank i is a box-node of box $\mathcal{B}$ with an edge to (j, B) and let (m, A) be the discharged node. Let $(\vec{q}, \Gamma)$ be the nodes that can be reached from $\mathcal{B}$ with one edge.



Let F be the graph consisting of box $\mathcal{B}$ with box-node n and all nodes that are reachable from $\mathcal{B}$ within one step. By induction we conclude that $\mathcal{V}(G)$ is $\mathcal{J}(G \setminus \overline{\mathcal{B}})$ glued to $\mathcal{I}(F)$. By Lemma 5.8.3 we are done.

5.9 Preservation of Reduction

We will show that the transformations involved in the process of cut-elimination of SGs can be mimicked by reductions on their context net translations. Of course this implies that it can be mimicked by reductions on the result of the direct translation of Section 5.5 too, and indeed on any translation that is an extension of the translation via context nets.

The notion of reduction on proof structures can be extended to reduction on context structures without any problem. We will use the following reduction rules: Ax-cut, $\parr$-$\otimes$, d-b, c-b and b-b. (See Section 5.2.)

Theorem 5.9.1 *The class of closed one-liners is closed under reduction.*

Proof Suppose C is a closed one-liner with initial nodes Γ and terminal nodes Δ. We can extend this to a proof net P with terminal nodes $\Delta, \bigotimes \Gamma^\perp$ (Figure

5.16). By reducing C to C' we obtain a proof structure P', which is a proof net, because the class of proof nets is closed under reduction. Remark that reduction rules preserve initial and terminal nodes. Therefore, C' is a closed one-liner.

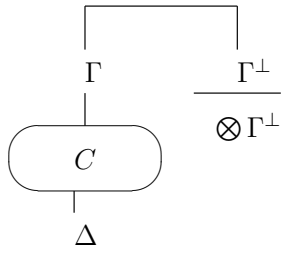
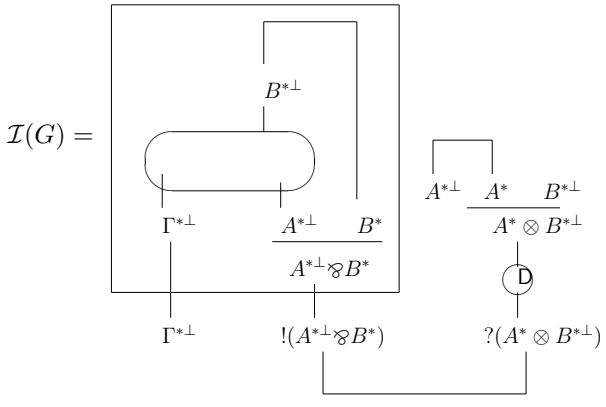


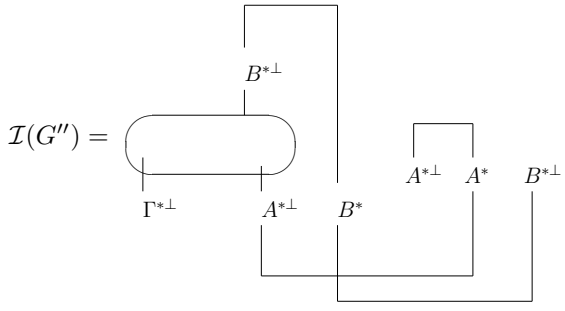
Figure 5.16: Closed one-liner C extended to a proof net.

Theorem 5.9.2 *If G' is obtained from G by either elimination of a safe cut, an unsharing step, a repeat-elimination step, or an incorporation step, then $\mathcal{I}(G')$ can be obtained from $\mathcal{I}(G)$ by one or more reductions of context nets.*

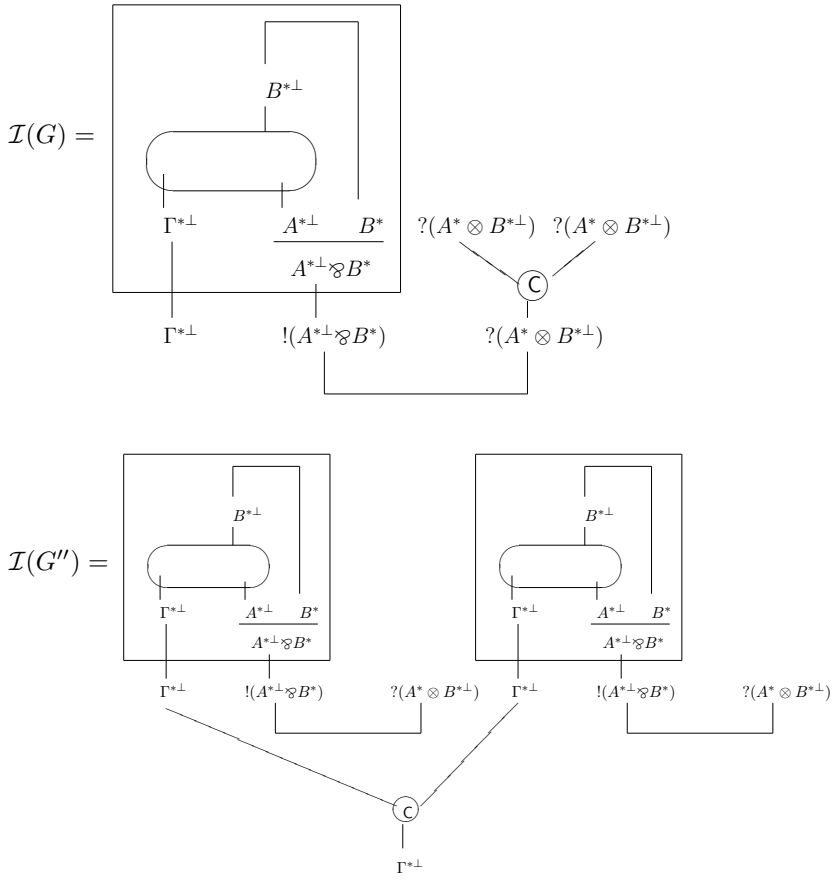
Proof

- If G' is obtained from G by eliminating a safe cut, then there exists a deduction net F such that $\mathcal{I}(G) \twoheadrightarrow_{d-b} F \twoheadrightarrow_{⅋-⊗} \mathcal{I}(G'')$.

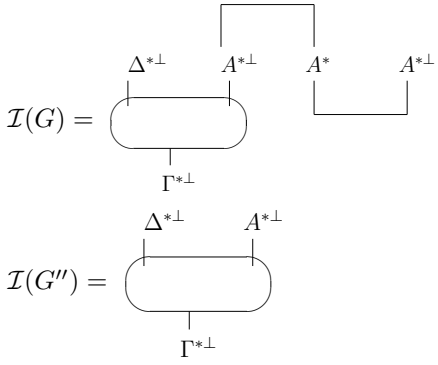




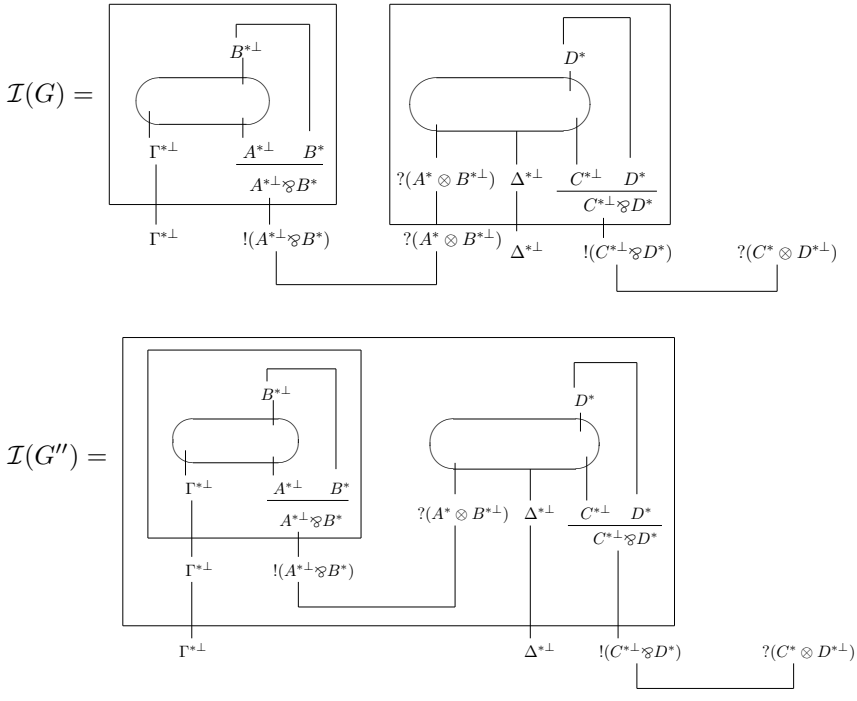
- If G' is obtained from G by unsharing, then $\mathcal{I}(G) \rightarrow_{c-b} \mathcal{I}(G')$.



- If G' is obtained from G by a repeat elimination, then $\mathcal{I}(G) \twoheadrightarrow_{Ax-cut} \mathcal{I}(G')$.



- If G' is obtained from G by incorporation, then $\mathcal{I}(G) \twoheadrightarrow_{b-b} \mathcal{I}(G')$.



5.10 Remarks concerning Strong Normalisation

From this correspondence of reduction follows again strong normalisation for the process of cut-elimination of SGs. However, the result that follows from Theorem 5.9.2 is stronger than the one mentioned in Section 5.3. To make this precise, let us identify three ways of applying the transformations:

Definition 5.10.1 *The process of* strict *cut-elimination, the process of* directed *cut elimination, and the process of* local *cut-elimination are respectively defined as follows:*

Strict *The cuts are eliminated in a strictly one by one fashion: Once started to make a cut safe, it is not allowed to start working on another cut, before the former has been eliminated. Every single cut is eliminated according to a certain strategy.*

Directed *The cuts are eliminated in a mixed way: Transformations concerning one cut may be interleaved with transformations concerning another one. However, all transformations we do are towards eliminating a cut.*

Local *Safe cut-elimination, repeat-elimination, incorporation and unsharing are seen as local transformations. So a repeat-elimination, an incorporation (on a box), or an unsharing (of boxes) may be done even when they are not necessary to make a cut safe. Care should be taken that the result is an* SG*.*

Note that strong normalisation for the process of local cut-elimination implies strong normalisation for the process of directed cut-elimination: The process of directed cut-elimination can be seen as a *strategy* for local cut-elimination (although the normal forms are not the same). Similarly, strong normalisation for the process of directed cut-elimination implies strong normalisation for the process of strict cut-elimination.

We can now reformulate the strong normalisation result mentioned in Section 5.3.

Theorem 5.10.2 *The process of strict cut-elimination on* SG*s is strongly normalising.*

Because Theorem 5.9.2 considers the transformations in a local fashion, we can conclude the following two stronger versions of strong normalisation for the process of cut-elimination on SGs.

Theorem 5.10.3 *The process of directed cut-elimination on* SG*s is strongly normalising.*

Theorem 5.10.4 *The process of local cut-elimination on* SG*s is strongly normalising.*

Chapter 6

Confluence

6.1 Introduction

In this chapter it is shown that the process of cut-elimination on SGs (Deduction Graphs with explicit sharing) is confluent. That is to say, that every order in which the cuts are eliminated leads to the same graph without cuts.

First, for every transformation (safe-cut elimination, repeat-elimination, incorporation, and unsharing) we fix some *special nodes and boxes*. Then we show that when an SG allows several transformations, and the special nodes and boxes do not overlap, then the transformations are independent: any order of applying transformations leads to the same result.

Furthermore, we show that when the special nodes and boxes of two transformations overlap, then applying either one, we can still obtain a common reduct.

This result is also known as *Weak Church-Rosser* (WCR) for SGs. It may be good to be explicit about what we prove exactly, because WCR may have various meanings, depending on how liberal the transformations may be applied. Let us therefore repeat Definition 5.10.1:

Definition 6.1.1 *The process of* strict *cut-elimination, the process of* directed *cut elimination, and the process of* local *cut-elimination are respectively defined as follows:*

Strict *The cuts are eliminated in a strictly one by one fashion: Once started to make a cut safe, it is not allowed to start working on another cut, before the former has been eliminated. Every single cut is eliminated according to a certain strategy.*

Directed *The cuts are eliminated in a mixed way: Transformations concerning one cut may be interleaved with transformations concerning another one. However, all transformations we do are towards eliminating a cut.*

Local *Safe cut-elimination, repeat-elimination, incorporation and unsharing are seen as local transformations. So a repeat-elimination, an incorporation*

(on a box), or an unsharing (of boxes) may be done even when they are not necessary to make a cut safe. Care should be taken that the result is an SG*.*

We start proving WCR for the process of directed cut-elimination. This means that all transformations that have to be done to arrive at the common SG, have to be of this class too: They have to be a safe cut-elimination, or they are part of the process to make a cut safe.

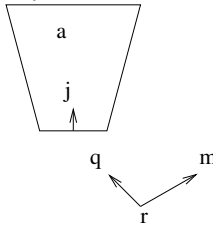
Then we extend the result to WCR for the local cut-elimination. This means that we have to examine some more cases, but also that the transformations applied to arrive at a common SG may be of a more general form.

Knowing that SGs are strongly normalising, we may conclude that SGs are confluent.

6.2 Weak Church-Rosser

Definition 6.2.1 (Special nodes and boxes) *The following nodes and boxes that are involved in a transformation, we call* special*:*

- *Safe cut-elimination:*

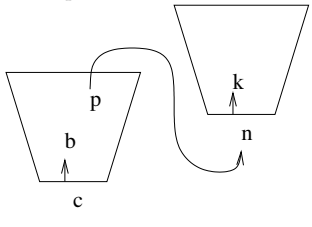


The special nodes are the box-node (q) of the box that disappears during elimination, the node it has an edge to (j), the A*-node in the box (a), the* E*-node of the cut (r), and the node it has an edge to besides q (m). The special box is the box that disappears at the elimination.*

- *Repeat-elimination:*

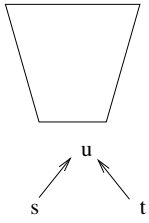
The special nodes are the node that disappears during elimination (e), and the node it has an edge to (d). There are no special boxes.

- *Incorporation:*



The special nodes are the box-node of the box that gets incorporated (n), the node it points to (k), and the node that points to it (p); the box-node of the deepest box that contains p (c), and the node it has an edge to (b). The special boxes are the box of n and the box of c.

- *Unsharing:*



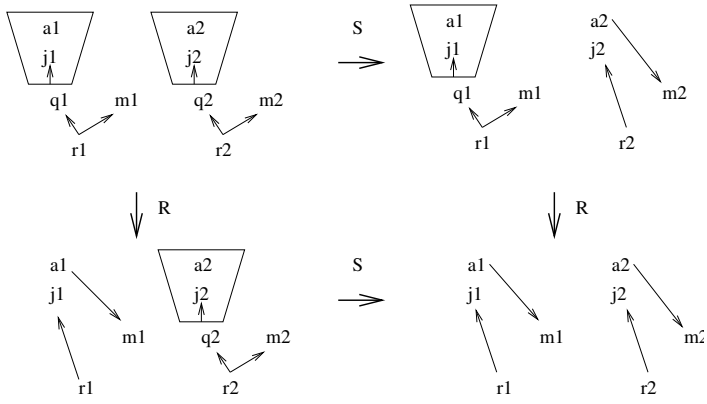
The special nodes are the S*-nodes involved in the unsharing (s and t), and the box-node they have an edge to (u). The special box is the box that gets copied.*

The names that are used for the nodes in this definition, will be used in the proofs of the following theorems.

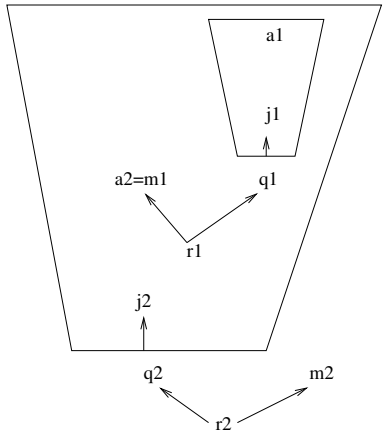
Lemma 6.2.2 *Let G be an* SG *and let $G \Rightarrow_R G'$ and let $G \Rightarrow_S G''$, where $\Longrightarrow_R$ and $\Longrightarrow_S$ are both safe cut-eliminations. Then there exists an* SG *F such that $G' \Rightarrow^* F$ and $G'' \Rightarrow^* F$.*

Proof Suppose $\Longrightarrow_R$ and $\Longrightarrow_S$ are different cut-eliminations. We distinguish cases according to the relative positions of the special nodes and boxes of $\Longrightarrow_R$ and $\Longrightarrow_S$.

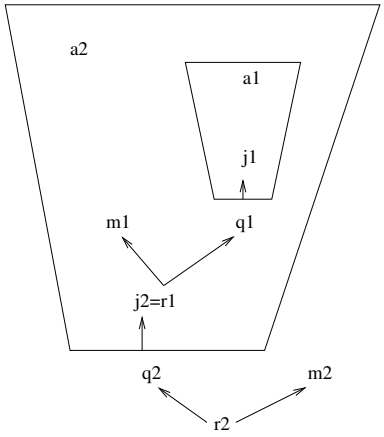
1. **Case $\{a_1, j_1, m_1, q_1, r_1\} \cap \{a_2, j_2, m_2, q_2, r_2\} = \emptyset$.** When we eliminate the safe cut indexed by 1, then we get G_1: the graph G without node q_1, without the edges $n_1 \longrightarrow\!\triangleright\, j_1$, $r_1 \longrightarrow\!\triangleright\, m_1$, and without the box of which q_1 is the box-node in G, and with the additional edges $p_1 \longrightarrow\!\triangleright\, j_1$ and $a_1 \rightarrow m_1$. So the cut indexed 2 is still there, and because there are no other edges to q_2 then already present in G, it is even a safe cut. Let G_{12} be the graph G_1 after eliminating the cut indexed 2. Similarly we define G_{21}, the graph G after first eliminating safe cut 2 and eliminating safe cut 1 afterwards. It is easy to see that $G_{12} = G_{21}$.



2. **Case** $a_2 = m_1$**.** When we do a cut-elimination on the cut marked 1 first, the nodes marked 2 are still part of a safe cut. On the other hand, when we do a cut-elimination on the nodes marked 2 first, the nodes marked 1 are still part of a safe cut. Furthermore, eliminating first cut 1 and then cut 2 yields the same result as eliminating first cut 2 and then cut 1. The case that the node r_2 is in several boxes that do not contain m_2, is similar.

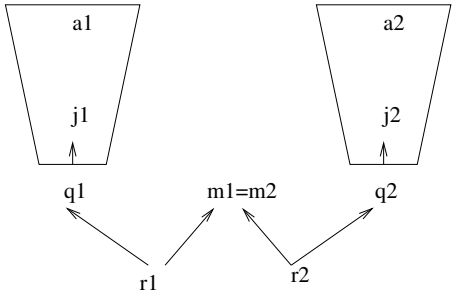


3. **Case** $j_2 = r_1$**.** When we do a cut-elimination on the cut marked 1 first, the nodes marked 2 are still part of a safe cut. On the other hand, when we do a cut-elimination on the nodes marked 2 first, the nodes marked 1 are still part of a safe cut. Furthermore, eliminating first cut 1 and then cut 2 yields the same result as eliminating first cut 2 and then cut 1. The case that the nodes r_1 and r_2 are in several boxes that do not contain m_1 and m_2 respectively, is similar.

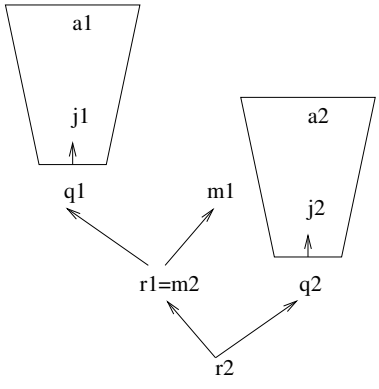


When additionally $a_2 = m_1$, the same argument applies.

4. **Case** $m_1 = m_2$**.** The two eliminations are interchangeable.



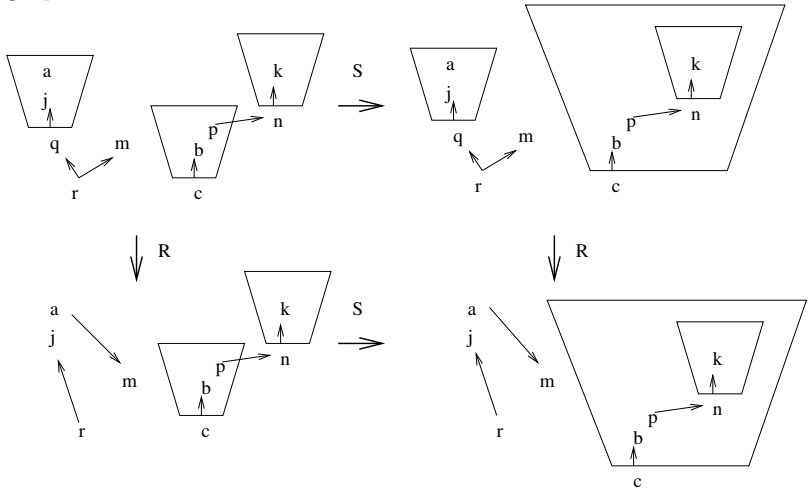
5. **Case** $r_1 = m_2$**.** The two eliminations are interchangeable.



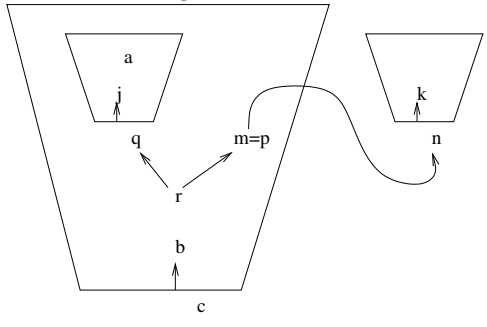
Lemma 6.2.3 *Let G be an* SG *and let $G \Rightarrow_R G'$ and let $G \Rightarrow_S G''$, where $\Rightarrow_R$ is a safe cut-elimination and $\Rightarrow_S$ is an incorporation. Then there exists an* SG *F such that $G' \Rightarrow^* F$ and $G'' \Rightarrow^* F$.*

Proof We distinguish cases according to the relative positions of the special nodes and boxes of $\Rightarrow_R$ and $\Rightarrow_S$.

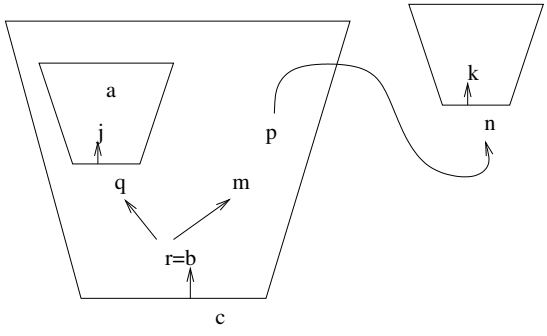
1. **Case** $\{a, j, q, m, r\} \cap \{k, n, b, c, p\} = \emptyset$ **and** $p \notin \mathcal{B}_q$**.** Let $\mathcal{C}$ be the largest box that contains p, but does not contain n. Let G_R be the graph after cut-elimination, that is G without the node q, without box $\mathcal{B}_q$, and without the edge $q \longrightarrow\!\triangleright\, j$, and with the additional edges $r \longrightarrow\!\triangleright\, q$ and $a \longrightarrow\!\triangleright\, m$. Then the edge $p \longrightarrow\!\triangleright\, n$ is still part of a cut-sequence, and this edge from n is still the only incoming edge of p. Moreover, the largest box containing p, but not containing n, is $\mathcal{C}$. Let G_{RS} be G_R after incorporation: so G_{RS} is G_R, with the difference that $\mathcal{B}_n$ is one level deeper. When we take G and do incorporation first, we get G_S: the graph G with box $\mathcal{B}_n$ one level deeper, so inside $\mathcal{C}$. This does not influence the safe cut. Let G_{SR} be the graph after cut-elimination. Then $G_{SR} = G_{RS}$.



2. **Case** $m = p$**.** In the following picture, p is part of a cut-sequence, ending in an arrow-elimination on a principle formula with the same label as n. Because r is the conclusion of an arrow-elimination on a principle formula with the label of q, this is not the cut of which n is a part. So, after cut-elimination, incorporation still applies. The other way around, if we do first incorporation, this leaves the safe cut unchanged. Moreover, eliminating first the safe cut and then the incorporation, yields the same result as doing the incorporation first, and the safe cut-elimination afterwards. Note that c might be in several boxes that do not contain n.

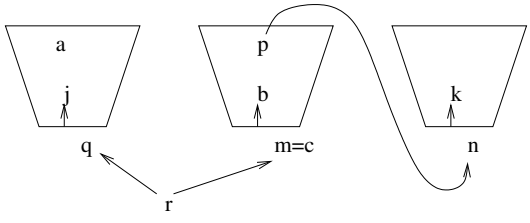


3. **Case** $r = b$**.** This case is similar to the former. Still m might be part of the cut-sequence containing p, so again it is necessary to check that r is not part of this cut-sequence. Another possibility is that p is at a greater depth than m. This does not change the situation in a significant way.

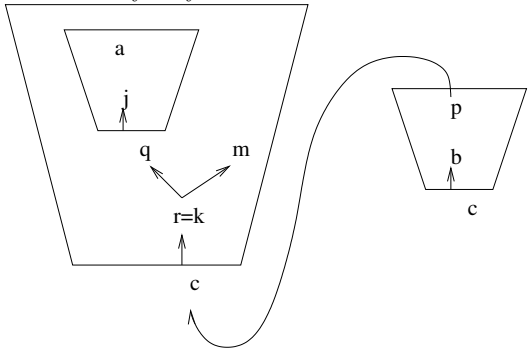


4. **Case both** $m = p$ **and** $r = b$. The argument is similar.

5. **Case** $m = c$**.** Here, the box of m is not the box that disappears during safe cut-elimination. Again the order of eliminating a safe cut and incorporating a box can be interchanged.

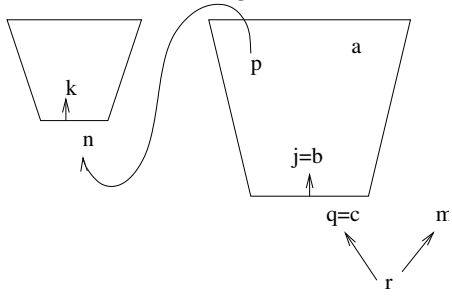


6. **Case** $r = k$**.** The safe cut and the incorporation do not influence each other in any way.

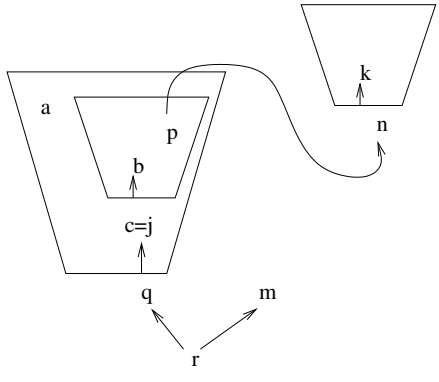


7. **Case** $j = b$ **and** $q = c$**.** In the following picture, the safe cut elimination does influence the incorporation. Suppose that n is at the same depth as q, and p is at the same depth as j. After cut-elimination, incorporation does not apply anymore. However, when we first do incorporation and then safe cut-elimination, the result is actually the same as doing only

safe cut-elimination. Suppose that n is at the same depth as q, but p is at a greater depth than j. Then after cut elimination, incorporation still applies. However, to get the same result as first incorporating the box, and then eliminating the safe cut, one should not apply it. Suppose that n is not at the same depth as q. Then after eliminating the safe cut, incorporation still applies and first eliminating the safe cut and then incorporating the box yields the same result as first incorporating the box and then eliminating the safe cut.



8. **Case $c = j$.** Suppose n is at the same depth as q. After cut-elimination, incorporation still applies. However, to get the same result as when first incorporating the box, and then eliminating the safe cut, one should not apply it. Suppose n is at the same depth as c. Then the safe cut-elimination and the incorporation are interchangeable. When there is a box that contains q, but does no contain n, then too the incorporation and the safe cut elimination are interchangeable.



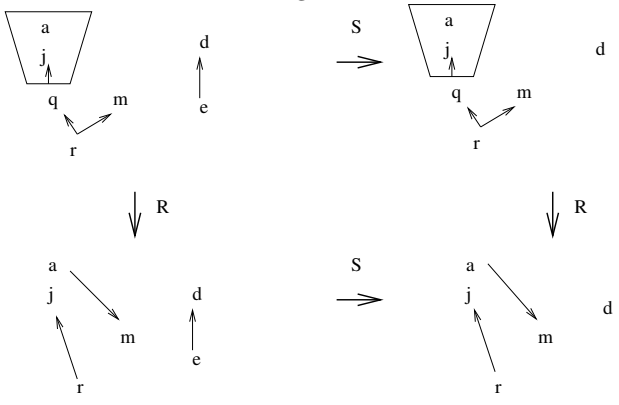
9. **Case $p \in \mathcal{B}_q$, but $\mathcal{B}_q$ is not the deepest box containing p,** and we are not in one of the previous situations. Suppose $\mathcal{B}_q$ is the largest box containing p, but not containing n. Then again, eliminating the safe cut yields the same result as first incorporating the box and then eliminating the safe cut. Suppose $\mathcal{B}_q$ is not the largest box containing p, then the incorporation and the elimination are interchangeable.

10. **Cases $b = p$ and $a = j$.** These do not change any of the above situations in any significant way (when applicable).

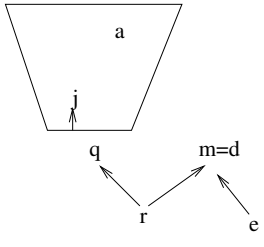
Lemma 6.2.4 *Let G be an* SG *and let $G \Rightarrow_R G'$ and let $G \Rightarrow_S G''$, where $\Rightarrow_R$ is a safe cut-elimination and $\Rightarrow_S$ is a repeat-elimination. Then there exists an* SG *F such that $G' \Rightarrow^* F$ and $G'' \Rightarrow^* F$.*

Proof We distinguish cases according to the relative positions of the special nodes and boxes of $\Rightarrow_R$ and $\Rightarrow_S$.

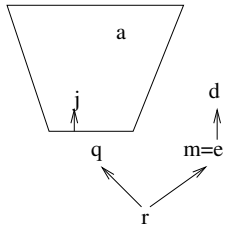
1. **Case $\{a, j, q, r, m\} \cap \{d, e\} = \emptyset$.** Then, after cut-elimination, $e \longrightarrow\!\triangleright\, d$ is still part of a cut-sequence. On the other hand, eliminating the repeat first does not influence the safe cut. The safe cut-elimination and the repeat-elimination are interchangeable.



2. **Case $m = d$.** Again we see that the cut-sequence of which e is part, has nothing to do with r. Therefore, eliminating the safe cut first does not interfere with the applicability of the repeat-elimination. Moreover, it yields the same result as when the repeat-elimination is handled first and the safe cut-elimination only afterwards.



3. **Case m=c**. Starting with safe cut-elimination, there will be an edge $a \longrightarrow\!\triangleright\, m$. After repeat-elimination this is replaced by an edge $a \longrightarrow\!\triangleright\, d$. When we do repeat elimination first, instead of an edge $r \longrightarrow\!\triangleright\, m$, we get the edge $r \longrightarrow\!\triangleright\, d$. Eliminating subsequently the safe cut produces an edge $a \longrightarrow\!\triangleright\, d$.

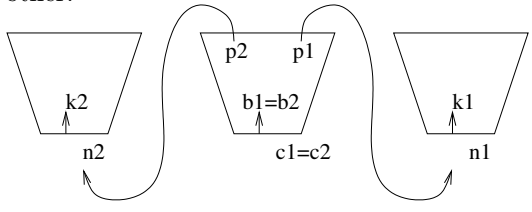


Lemma 6.2.5 *Let G be an* SG *and let $G \Rightarrow_R G'$ and let $G \Rightarrow_S G''$, where $\Rightarrow_R$ and $\Rightarrow_S$ are both incorporations. Then there exists an* SG *F such that $G' \Rightarrow^* F$ and $G'' \Rightarrow^* F$.*

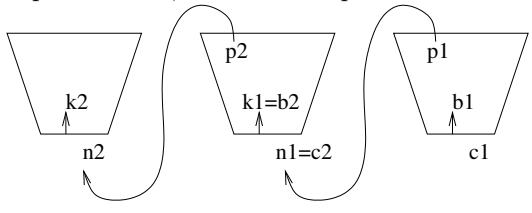
Proof Suppose $\longrightarrow_R$ and $\longrightarrow_S$ are different cut-eliminations. We distinguish cases according to the relative positions of the special nodes and boxes of $\longrightarrow_R$ and $\longrightarrow_S$.

1. **Case $\{k_1, n_1, p_1, b_1, c\} \cap \{k_2, n_2, p_2, b_2, c_2\} = \emptyset$, and $p_1 \notin \mathcal{B}_{n_2}$, $p_2 \notin \mathcal{B}_{n_1}$.** Let $\mathcal{C}_1$ be the largest box containing p_1, but not containing n_1; let $\mathcal{C}_2$ be the largest box containing p_2. After the incorporation indexed by 1, $\mathcal{B}_{n_1}$ is one level deeper. Incorporation 2 is still applicable, and $\mathcal{C}_2$ is the largest box that contains p_2, but doesn't contain n_2. Note that the incorporations are interchangeable.

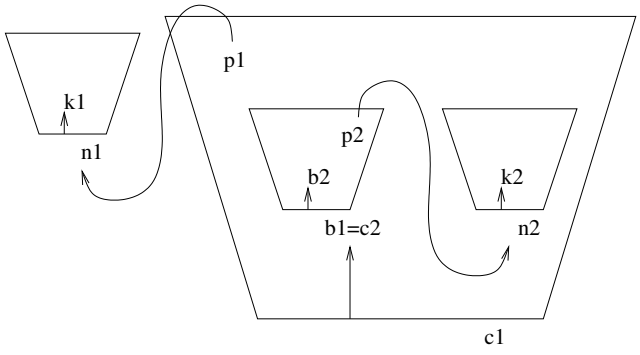
2. **Case $b_1 = b_2$ and $c_1 = c_2$.** The two incorporations do not influence each other.



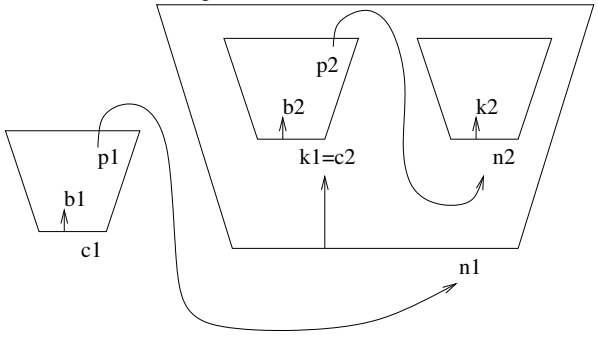
3. **Case $k_1 = b_2$ and $n_1 = c_2$.** Suppose n_1 and n_2 are at the same depth. Then, after the incorporation indexed by 1, one has to incorporate twice the box of n_2 to get the same result as when the incorporation indexed by 1 followed the incorporation indexed by 2. In case n_1 is at a greater depth than n_2, the two incorporations are interchangeable.



4. **Case $b_1 = c_2$.** The two incorporations do not influence each other.



5. **Case** $k_1 = c_2$**.** Suppose $n_2 \notin \mathcal{B}_{n_1}$. Then first incorporating $\mathcal{B}_{n_2}$ and then $\mathcal{B}_{n_1}$ corresponds to first incorporating $\mathcal{B}_{n_1}$ and then incorporating $\mathcal{B}_{n_2}$ twice. If $n_2 \in \mathcal{B}_{n_1}$, the two incorporations are interchangeable.



6. **Case** $p_2 \in \mathcal{B}_{n_1}$**, but** $k_1 \neq c_2$**.** This is similar to the previous case.

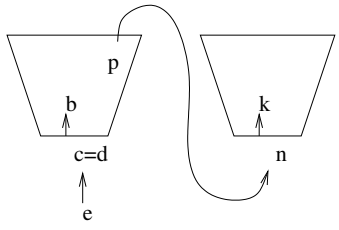
7. **Cases** $p_1 = b_1$ **or** $p_2 = b_2$**.** These do not change the above situations in any significant way (when applicable).

Lemma 6.2.6 *Let G be an* SG *and let $G \Rightarrow_R G'$ and let $G \Rightarrow_S G''$, where $\Rightarrow_R$ is an incorporation and $\Rightarrow_S$ is a repeat-elimination. Then there exists an* SG *F such that $G' \Rightarrow^* F$ and $G'' \Rightarrow^* F$.*

Proof We distinguish cases according to the relative positions of the special nodes and boxes of $\Rightarrow_R$ and $\Rightarrow_S$.

1. **Case** $\{c, b, p, n, k\} \cap \{d, e\} = \emptyset$**.** Then, after repeat-elimination, the incorporation is still applicable. Similarly: After first incorporating the box, the repeat-elimination is still applicable. We have that $G_{RS} = G_{SR}$.

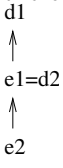
2. **Case** $c = d$**.** The incorporation and the repeat-elimination are interchangeable.



Lemma 6.2.7 *Let G be an* SG *and let $G \Rightarrow_R G'$ and let $G \Rightarrow_S G''$, where $\Rightarrow_R$ and $\Rightarrow_S$ are both repeat-elimination. Then there exists an* SG *F such that $G' \Rightarrow^* F$ and $G'' \Rightarrow^* F$.*

Proof Suppose $\Rightarrow_R$ and $\Rightarrow_S$ are different repeat-eliminations. We distinguish cases according to the relative positions of the special nodes and boxes of $\Rightarrow_R$ and $\Rightarrow_S$.

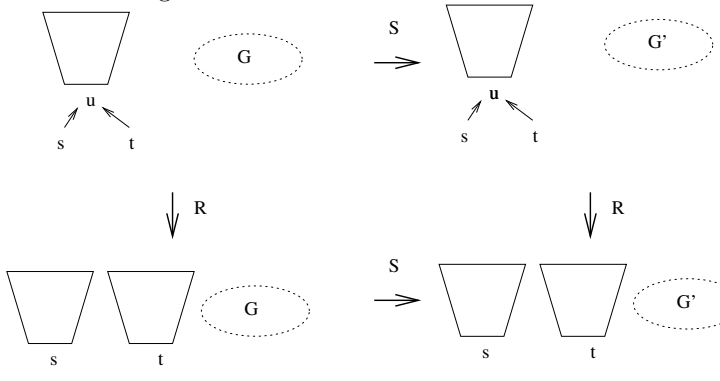
1. $\{d_1, e_1\} \cap \{d_2, e_2\} = \emptyset$**.** Then the repeat-eliminations are interchangeable.
2. **Case** $e_1 = d_2$**.** After two repeat-eliminations only d_1 is left, irrespectively the order.



Lemma 6.2.8 *Let G be an* SG *and let $G \Rightarrow_R G'$ and let $G \Rightarrow_S G''$, where $\Rightarrow_R$ is an unsharing. Then there exists an* SG *F such that $G' \Rightarrow^* F$ and $G'' \Rightarrow^* F$.*

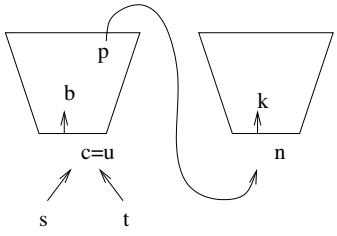
Proof

- Suppose all nodes, edges and boxes directly involved in $\Rightarrow_S$ are outside the box that gets duplicated during the unsharing. Then the transformations are interchangeable.

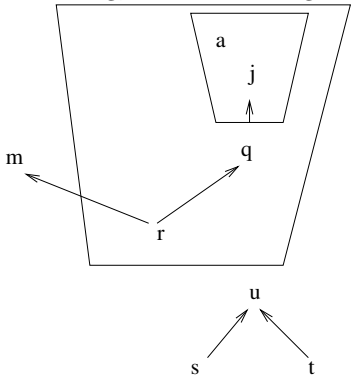


- Suppose $\Rightarrow_S$ is incorporation. Suppose we start with the unsharing of box c. Then we get two edges to n: $n' \rightarrowtriangle n$ and $n'' \rightarrowtriangle n$, so incorporation is not

immediately possible. One should do an additional unsharing, followed by two incorporations, to obtain the same result as incorporating the box first, and unsharing only afterwards. (A similar argument applies, when u is a box-node of a box containing p, but not the deepest one.)



- Suppose $\longrightarrow_S$ is safe cut-elimination. Eliminating the safe cut first, and unsharing the box only later, yields the same result as starting with the unsharing, and eliminating a safe cut twice afterwards.



- Suppose all nodes, edges and boxes directly involved in $\Rightarrow_S$ are inside the box that gets duplicated. Then after the unsharing, we have to do $\longrightarrow_S$ twice to get the same result as first doing $\longrightarrow_S$ and then unsharing.

Theorem 6.2.9 (Weak Church-Rosser; directed) *Suppose G is an* `SG` *and $G \Rightarrow_R G'$ and $G \Rightarrow_S G''$, where $\Rightarrow_R$, $\Rightarrow_S$ are transformations of the process of directed cut-elimination. Then there exists an* `SG` *F such that $G' \Rightarrow^* F$ and $G'' \Rightarrow^* F$.*

Proof Immediately by the Theorems 6.2.2, 6.2.3, 6.2.4, 6.2.5, 6.2.6, 6.2.7 and 6.2.8.

It might be difficult to find a cut, as the arrow-introduction and the arrow-elimination might be separated by many repeat-nodes, sharing-nodes, and boxes. It might therefore be convenient to, for example, just do incorporation, whether this will eventually lead to a safe cut or not. Therefore it is useful to have WCR for the process of local cut-elimination too.

Theorem 6.2.10 (Weak Church-Rosser; local) *Suppose G is an* SG *and $G \to_R G'$ and $G \to_S G''$, where $\to_R$, $\to_S$ are transformation of the process of local cut-elimination. Then there exists an* SG *F such that $G' \to^* F$ and $G'' \to^* F$.*

Proof We have to complete the following lemmas with the following cases.

- Lemma 6.2.4 (safe cut-elimination and repeat-elimination). We also have the cases $a = d$ and $j = e$. It is easy to see that in those cases the reductions are interchangeable.
- Lemma 6.2.5 (two incorporations). We have the additional cases $p_1 = k_2$ and $p_1 = b_1$. The former is similar to the second case of the lemma, as $p_1 = k_2$ implies that $k_2 = b_1$ and $n_2 = c_1$. The latter is similar to the third case of the lemma, as $p_1 = b_2$ implies that $b_1 = b_2$.
- Lemma 6.2.6 (incorporation and repeat-elimination). We also have the cases $p = d$, $k = e$ and $b = e$. It is easy to see that in those cases the reductions are interchangeable.

Theorem 6.2.11 *Both the process of directed cut-elimination and the process of local cut-elimination on* SG*s are confluent.*

Proof By Newman's Lemma, we may conclude from Strong Normalisation (Theorems 5.10.3 and 5.10.4) and Weak Church Rosser (Theorems 6.2.9 and 6.2.10) that the transformations on SGs are confluent.

6.3 Optimal reductions, closed reductions and strategies

6.3.1 Sharing Graphs and Optimal Reductions

Reduction of a simply typed λ-term often leads to duplication of work. It is therefore natural to search for an *optimal* reduction strategy to reduce these terms. For *reduction strategy* we take the following definition (see [83] for a somewhat more general definition):

Definition 6.3.1 *A* reduction strategy *for a rewriting system (of either terms or graphs) is a subsystem with the same objects and normal forms.*

Definition 6.3.2 *A reduction strategy is called* optimal*, when it reduces every term or graph in a minimal amount of steps to a normal form. That is to say: any other reduction path is at least as long.*

However, there is no recursive optimal strategy for the λ-calculus with (one-step) β-reduction [11] [9].

The situation changes when we do not consider one-step β-reduction, but *multi*-step β-reduction instead. That is: one reduction step may be comprised of several β-reductions (see also [83]).

In that case there exists a recursive optimal strategy, but it is still very complex. This follows from Statman's result [81] that it cannot be Kalmar-elementary decided whether two terms reduce to the same normal form. This entails that there exists no Kalmar-elementary optimal strategy for the simply typed λ-calculus with respect to multi-step reduction (see [6]).

Lévy [55] has defined such a recursive optimal multi-step reduction strategy. He defines *families* of redexes in a term and he introduces a reduction which contracts a whole family of β-reductions in a single reduction step. For this reduction a leftmost-outermost strategy, i.e. contract a family that contains a leftmost-outermost redex, is optimal.

It should however be remarked that the redexes of a family are in general spread over a term, and that Lévy has not explained how family reduction can be considered as one single step in practice: In his approach a family reduction is not local. Lamping [53] then visualised family reduction in the so-called *sharing graphs*, which equate all redexes of the same family: Every redex of a family is represented by one and the same redex (see also [4]).

A sharing graph is a representation of a λ-term that allows not only sharing of subterms, but also sharing of subcontexts. In this they differ from DGs and SGs. Also the duplication mechanism is more subtle and refined than the one of DGs, SGs and proof nets, in which duplication of parts of the term is done box-wise. Sharing graphs do not contain boxes, but they have special nodes (*brackets* and *croissants*), which can be seen as borders between nodes that should be thought of as belonging to "a box", and nodes that are outside of it. Duplication is then done by moving the special nodes towards each other, and copying the nodes that a special node crosses in the process one by one. When two special nodes bordering the "same box" meet, they disappear.

So, these special nodes allow a distributed notion of box. And this notion makes all steps local, parallel, and asynchronous in the sense of [52].

From [6] it follows that, since family reduction is local, the sharing mechanism has to be very complex.

6.3.2 Innermost Reduction

With the coarse duplication mechanism SGs have, it is very unlikely that there exists an recursive optimal strategy for them, independent of whether we count all transformation steps, or only the safe cut-eliminations.

On the other hand, there are systems known that do have such an optimal strategy. For Orthogonal Term Rewriting Systems (OTRSs) [47], for example, it is innermost essential normalising reduction (*internal needed strategy* in [83]). However this strategy is in general not computable, as neededness is not.

Similarly, a kind of "innermost" reduction strategy might be optimal for SGs. Innermost seems the most likely candidate, because any strategy that duplicates redexes is surely not optimal. This strategy follows from a careful study of the

WCR proof (Theorem 6.2.9). Although after doing either one of the reductions of a critical pair we can still obtain a common reduct, the length of the path to this common reduct depends on our choice. One could hope that choosing the best option locally (the reduction leads to the shortest path to a common reduct) at every stage, may eventually lead to the best choice globally; that is to an optimal path to the normal form.

Concretely, this would be the following strategy for SGs:

Definition 6.3.3

1. *Do all repeat-eliminations.*
2. *Eliminate the safe cut lowest in the box-topological ordering (and start again at point 1).*
3. *Incorporate the box with the lowest possible box-node in the box-topological ordering (and start again at point 1).*
4. *Unshare the box with the lowest possible box-node (and start again at point 1).*

Here we consider the transformation steps as belonging to the process of directed cut-elimination. (Definition 6.1.1). For example, the first command of the strategy should be read as: "Do all repeat-eliminations *that are part of the process of making a cut safe.*"

It turns out, however, that this strategy is not optimal, which is shown by the following example[1].

Example 6.3.4 *Normalising the graph of Figure 6.1 following the strategy of Definition 6.3.3 amounts to: A safe cut-elimination on the cut of the nodes* $8, 14, 15$*; an unsharing followed by the safe cut-elimination belonging to the* E*-nodes* 6 *and* 7*; a safe-cut-elimination on the cut of the nodes* $17, 18, 19$*; finally a repeat-elimination, an unsharing and two safe-cut-eliminations, destructing the box belonging to node* 18 *(and its copy). This is a total of ten transformation steps, of which six are safe cut-eliminations.*

A shorter approach would be to start with the safe cut-elimination on the cut of the nodes $17, 18, 19$*; then a repeat-elimination, an incorporation, and the safe cut-elimination destructing the box belonging to node* 18*; a safe cut-elimination on the cut of the nodes* $8, 14, 15$*; finally a repeat-elimination, and an unsharing followed by the safe cut-eliminations belonging to the* E*-nodes* 6 *and* 7*. This is a total of only nine transformation steps, of which just five are safe cut-eliminations.*

The difference can be made arbitrarily big by sharing $A \to B$ *in the box of box-node* 8 *arbitrarily often.*

Although this reduction strategy is not optimal in general, it might be optimal when we restrict our notion of reduction, similarly to Wadsworth's strategy and weak reduction [12].

[1] Personal communication Vincent van Oostrom.

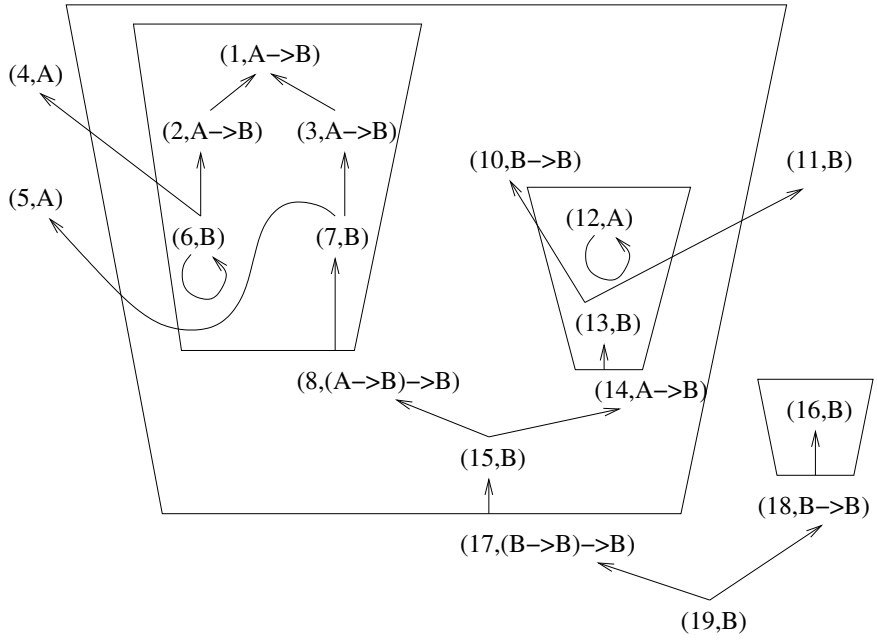


Figure 6.1: Innermost reduction is not optimal.

6.3.3 Closed Reductions

The consequences of this strategy (Definition 6.3.3) at the proof net side shows remarkable similarities to the idea of *closed reductions*.

The strategy of closed reductions [57] [58] for proof nets originates from an encoding of proof nets in interaction nets [51], a formalism that allows only local transformations and that is free from critical pairs. In short, the strategy of closed reductions prescribes that:

1. Reductions involving boxes (b-d, b-c, b-b, b-w) are only allowed when the box has just one terminal node ($!A$). In b-b reduction this concerns the box that gets incorporated.

2. b-c reduction is only allowed when the part of the proof net that gets copied is already in normal form.

When we consider deduction graphs without free nodes, requirement 3 of Definition 6.3.3 comes close to the first requirement of closed reductions for b-b reductions. Remark that it is not exactly the same, as Definition 6.3.3 does not prescribe to incorporate boxes that are not involved in the process of cut-elimination. Requirement 4 of Definition 6.3.3 corresponds to the second requirement of closed reductions as stated above.

The strategy of closed reductions for proof nets is quite efficient [58] [79].

Chapter 7

Universal Quantification

7.1 Introduction

In this chapter we extend DGs, the deduction graphs defined in Chapter 2, with first-order universal quantification. Originally, boxes were meant to border the scope of a local assumption, but now we also use boxes to border the scope of a quantifier. This extension can be carried through without much ado. This shows the robustness of the concept of deduction graphs.

In Section 7.2 we give the definition of these deduction graphs with universal quantification, called $\mathtt{DG_U}$s, starting from definitions for terms and formulas of first-order predicate logic. Section 7.3 extends the embedding results from Fitch style flag deduction and Gentzen-Prawitz style natural deduction. The process of cut-elimination is discussed in Section 7.4, followed by strong normalisation in Section 7.5. Finally, Section 7.6 compares $\mathtt{DG_U}$s with developments in proof nets.

7.2 Definition

Contrary to the language of first-order predicate logic for Gentzen-Prawitz style natural deduction [27] [70], we define the language Pred of first-order predicate logic with universal quantification and equality for deduction graphs to have two kinds of variables. The first kind, denoted by $u, v, w, \ldots$, are meant to be used as *free* variables. The second kind, $x, y, z, \ldots$, will only be used *bound*. The same idea is often used for the language of first-order predicate logic for Fitch style flag deductions [85]. The reason we define it this way is that it facilitates the process of cut-elimination.

We now define the terms, basic formulas, formulas and axioms of Pred.

Definition 7.2.1 *The* terms *of* Pred *are defined as follows:*

1. *Every variable u is a term.*

2. *For every n-ary function symbol* $\mathtt{F}$ *and every sequence* $t_1, t_2, \ldots, t_n$ *of terms,* $\mathtt{F}(t_1, t_2, \ldots, t_n)$ *is a term.*

3. *The constructions 1 and 2 are the only ways to get terms.*

Definition 7.2.2 *The* basic formulas *of* Pred *are defined as follows:*

1. *For every n-ary relation symbol* R *and every sequence* $t_1, t_2, \ldots, t_n$ *of terms,* $\mathtt{R}(t_1, t_2, \ldots, t_n)$ *is a basic formula;*

2. *For every two terms* s *and* t, $s = t$ *is a basic formula.*

3. *The constructions 1, 2 and 3 are the only ways to get basic formulas.*

Definition 7.2.3 *The* formulas *of* Pred *are defined as follows:*

1. *Every basic formula is a formula;*

2. *For every formula* φ *and all variables* x *and* u, $(\forall x.\varphi[x/u])$ *is a formula, under the proviso that* $\forall x$ *does not occur in* φ.

3. *For all formulas* φ *and* ψ, $(\varphi{\rightarrow}\psi)$ *is a formula.*

4. *The constructions 1, 2 and 3 are the only ways to get formulas.*

Furthermore, we adopt the following convention for the brackets in formula:

1. $\rightarrow$ is right-associative;

2. $\forall$ binds stronger than $\rightarrow$;

3. Outer brackets are not written, nor are any other brackets that do not contribute to our understanding of the formula.

So, for example, $\forall x.\varphi{\rightarrow}\psi{\rightarrow}\xi \equiv ((\forall x.\varphi){\rightarrow}(\psi{\rightarrow}\xi))$. Note, however, that $\forall x.\mathtt{P}(x){\rightarrow}\mathtt{Q}(x)$ can formaly only be understood as $(\forall x.(\mathtt{P}(x){\rightarrow}\mathtt{Q}(x)))$, because $((\forall x.\mathtt{P}(x)){\rightarrow}\mathtt{Q}(x))$ is not a formula. In these kind of cases we will write the inner brackets under the quantifier explicitely anyway, for the convention would otherwise lead us to misinterpret the formula.

Definition 7.2.4 *The* equivalence axioms *are:*

1. $\forall x.x = x$ *(identity);*

2. $\forall x.\forall y.(x = y{\rightarrow}y = x)$ *(symmetry);*

3. $\forall x.\forall y.\forall z.(x = y{\rightarrow}y = z{\rightarrow}x = z)$ *(transitivity).*

4. *For every n-ary relation symbol* $\mathtt{R}$*:*
$\forall x_1.\ldots.\forall x_n.\forall y_1.\ldots.\forall y_n.$
$(x_1 = y_1{\rightarrow}\ldots{\rightarrow}x_n = y_n{\rightarrow}\mathtt{R}(x_1, \ldots, x_n){\rightarrow}\mathtt{R}(y_1, \ldots, y_n))$ *(relation axioms);*

5. *For every n-ary function symbol* $\mathtt{F}$*:*
   $\forall x_1 \ldots \forall x_n . \forall y_1 \ldots \forall y_n .$
   $(x_1 = y_1 \rightarrow \ldots \rightarrow x_n = y_n \rightarrow \mathtt{F}(x_1, \ldots, x_n) = \mathtt{F}(y_1, \ldots, y_n))$ *(function axioms);*

Because Pred deviates from the language of first-order predicate logic for Gentzen-Prawitz natural deduction, this also means that the $\forall$-introduction for deduction graphs cannot be similar to the $\forall$-introduction in the Gentzen-Prawitz formalism.

The $\forall$-introduction in Gentzen-Prawitz style natural deduction is as follows:

$$\dfrac{\dfrac{D}{\varphi}}{\forall x.\varphi}$$

Where x may not be free in the non-discharged assumptions of D. This means that x might be free in φ, although it is bound in $\forall x.\varphi$.

In deduction graphs we will introduce a new (bound) variable in the $\forall$-introduction step. The advantage is then, that a graph may still be correct, when we rename only free variables. We will use this later, in the process of cut-elimination.

Definition 7.2.5 *The collection of* deduction graphs for first-order universal quantification, $\mathtt{DG_U}$ *is the set of closed box directed graphs over* $\mathbb{N} \times \mathsf{Pred}$ *inductively defined as follows:*

DG *Every construction rule for deduction graphs for minimal proposition logic, except* **Join** *is a construction rule for* $\mathtt{DG_U}$*s.*

$\forall$-I *If* G *is a* $\mathtt{DG_U}$ *containing a node* (j, φ) *with no ingoing edges at top-level for some formula* φ *of* Pred*, then the graph* $G' := G$ *with*

- *A box* $\mathcal{B}$ *with box-node* $(n, \forall x.\varphi[x/u])$ *, not containing any nodes without outgoing edges, where we call* u *the* eigenvariable *of* $\mathcal{B}$ *when* u *occurs in* φ*,*
- *An edge from the box-node* $(n, \forall x.\varphi[x/u])$ *to* (j, φ)

is a $\mathtt{DG_U}$ *under the proviso that:*

- G' *is a well-formed closed box directed graph,*
- u *does not occur in the label of any node, that is not in* $\mathcal{B}$*;*

$\forall$-E *If* G *is a* $\mathtt{DG_U}$ *with a node* $(n, \forall x.\varphi)$ *at top-level for some formula* φ *of* Pred*, then the graph* $G' := G$ *with*

- *a node* $(p, \varphi[t/x])$ *where none of the variables of* t *is the eigenvariable of any box of* G*,*
- *an edge from* $(p, \varphi[t/x])$ *to* $(n, \forall x.\varphi)$

is a $\mathtt{DG_U}$.

Join$_{\mathtt{U}}$ *If G and G' are two $\mathtt{DG_U}$s then $G'' = G \cup G'$ is a $\mathtt{DG_U}$ under proviso that the eigenvariables of the boxes of G do not appear in the labels of the nodes of G' and vice versa.*

Example 7.2.6 *Let* $\mathtt{P}$ *and* $\mathtt{Q}$ *be unary predicate symbols of* Pred. *Figures 7.1 and 7.2 show examples of* $\mathtt{DG_U}$*s. We draw the border of a quantifier box as an interrupted line, to distinguish it from the implicational boxes.*

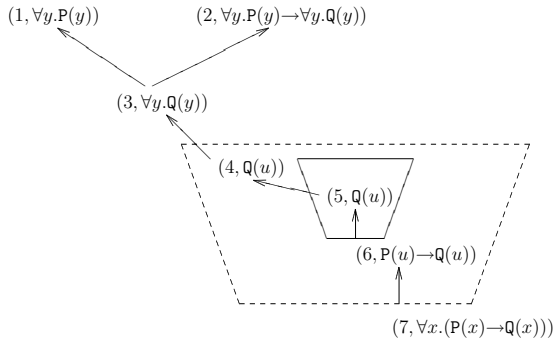


Figure 7.1: An example of an $\mathtt{DG_U}$.

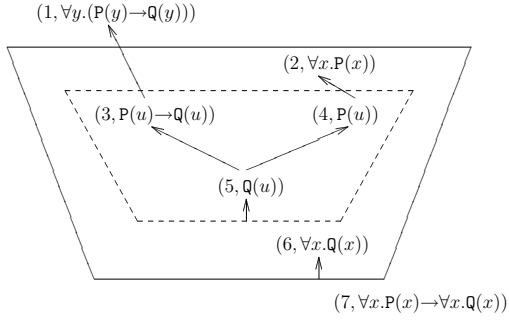


Figure 7.2: Another example of a $\mathtt{DG_U}$.

Lemma 7.2.7 *Let G be a $\mathtt{DG_U}$. Then for every variable u:*

1. *u occurs as eigenvariable of a box of G at most once;*
2. *If u is an eigenvariable of a box $\mathcal{B}$ of G, it does not occur in a label of a node outside $\mathcal{B}$.*

Proof The proof is by induction:

- Suppose G has been created by the **Axiom**-rule. Then no variable is eigenvariable of any box of G.
- Suppose u is an eigenvariable of exactly one box $\mathcal{B}$ of G and u does not occur in a label of any box outside $\mathcal{B}$. Suppose we obtain G' from G by applying the $\rightarrow$**-E**-rule, the $\rightarrow$**-I**-rule, or the **Repeat**-rule. Then u is still an eigenvariable of exactly one box $\mathcal{B}$ of G' and u does not occur in a label of a node outside $\mathcal{B}$. Moreover, these steps do not add any new eigenvariables.
- Suppose u is an eigenvariable of exactly one box $\mathcal{B}$ of G and u does not occur in a label of any box outside $\mathcal{B}$. Suppose G' is obtained from G by applying the $\forall$**-I**-rule, say the node added is $(n, \forall x.\varphi[x/v])$. Because u does not occur in a label outside $\mathcal{B}$, $u \neq v$ (or v does not occur in φ). So u is still an eigenvariable of exactly one box $\mathcal{B}$ of G' and u does not occur in a label of a node outside $\mathcal{B}$. This also means that v is the eigenvariable of at most one box of G'. The last requirement of the $\forall$**-I**-rule says that v does not occur outside the newly created box.
- Suppose u is an eigenvariable of exactly one box $\mathcal{B}$ of G and u does not occur in a label of any box outside $\mathcal{B}$. Suppose G' is obtained from G by applying the $\forall$**-E**-rule. In this step, no new boxes are created, so u is still the eigenvariable of at most one box. Let $(p, \varphi[t/x])$ be the added node in this step. Then u is not a variable of t. So u does not occur in a label of any node outside $\mathcal{B}$ in G'.
- Suppose G and G' are two $\mathtt{DG_U}$s, such that for every variable u: u is the eigenvariable of a box in G at most once and if it is the eigenvariable of a box $\mathcal{B}$ of G, then is does not occur in a label of a node outside $\mathcal{B}$. Suppose that the same statement holds when we replace G by G'. Suppose u is an eigenvariable of exactly one box $\mathcal{B}$ of G and u does not occur in a label of any box outside $\mathcal{B}$. Suppose F is obtained from G and G' by applying the $\mathbf{Join}_{\mathtt{U}}$-rule. Then u does not occur in G', neither as an eigenvariable of a box, nor anywhere outside a box. Hence u is still an eigenvariable of exactly one box $\mathcal{B}$ of F and u does not occur in a label of a node outside $\mathcal{B}$.

We will formulate a criterion to check relatively easily whether a given closed box directed graph is a $\mathtt{DG_U}$ (Lemma 7.2.10). As an important notion for $\mathtt{DG_U}$s is the eigenvariable of a box, we need a similar notion for general closed box directed graphs.

Definition 7.2.8 (Box-variable) *Let G be a closed box directed graph. We call a variable u a box-variable of $\mathcal{B}$, when u does not occur in the label of the box-node of $\mathcal{B}$, but it does occur in the label of the node that the box-node points to.*

Remark 7.2.9 *Note that for $\mathtt{DG_U}$s the notion of eigenvariable and the notion of box-variable coincides.*

Lemma 7.2.10 *G is a $\mathtt{DG_U}$ if and only if the following hold*

1. *If u is a box-variable of a box $\mathcal{B}$ of G, it does not occur in a label of a node outside $\mathcal{B}$.*
2. *G is a finite closed box directed graph.*
3. *There is a box-topological ordering $>$ of G.*
4. *Every node of G is of one of the following six types:*

   A *n has no outgoing edges.*

   $\rightarrow$-E *n has label B and has exactly two outgoing edges: one to a node $(m, A{\rightarrow}B)$ and one to a node (p, A), both within the scope of n.*

   $\rightarrow$-I *n is a box-node of a box $\mathcal{B}$ with label $A{\rightarrow}B$ and has exactly one outgoing edge, which is to a node (j, B) inside the box $\mathcal{B}$ (and not in any deeper boxes) with no other ingoing edges. All nodes inside the box without outgoing edges have label A.*

   R *n has label A and has exactly one outgoing edge, which is to a node (m, A) that is within the scope of n.*

   $\forall$-E *n has label $\varphi[t/x]$ for some formula φ, some term t and some variable x, and n has exactly one outgoing edge: to a node $(m, \forall x.\varphi)$ within the scope of n.*

   $\forall$-I *n is a box-node of a box $\mathcal{B}$ with label $\forall x.\varphi$ for some variable x and some formula φ, and has exactly one outgoing edge, which is to a node $(j, \varphi[u/x])$ inside the box $\mathcal{B}$ (and not in any deeper boxes) with no other ingoing edges. There are no nodes without outgoing edges in $\mathcal{B}$.*

Proof $\Leftarrow$:By induction on the number of nodes of G. Let $<$ be the box-topological order that is assumed to exist. Let n be the node that is maximal w.r.t. $<$. Then n must be on top-level. When we remove n, possibly including its box (if n is of type $\rightarrow$-I or $\forall$-I), we obtain a graph G' that again satisfies the properties listed in the lemma. By induction hypothesis we see that G' is a deduction graph. Suppose n is a node of type

A We obtain G by applying the Axiom-rule and the $\mathrm{Join_U}$-rule. Suppose u is an eigenvariable of a box of G', then u is a box-variable of a box of G and it does not occur outside the box. So especially it does not occur in the label of n.

$\rightarrow$-E We obtain G by applying the $\rightarrow$-E rule.

$\rightarrow$-I We obtain G by applying the $\rightarrow$-I rule.

R We obtain G by applying the Repeat-rule.

∀-E We obtain G by applying the ∀-E rule. Suppose u a box-variable of a box of G, then u does not occur outside the box, so especially it does not occur in the label of n.

∀-I We obtain G by applying the ∀-I rule. Because the variable u occurs at most once as box-variable in G (by requirement 1) and it does not occur outside the box of which it is the box-variable, the resulting graph is a correct $\mathtt{DG}_{\mathtt{U}}$.

⇒: By induction to the construction of G, using Lemma 7.2.7, statement 2.

7.3 Embeddings from Other Formalisms

7.3.1 Fitch style flag deduction

Fitch style flag deductions for first-order predicate logic is normally done using rules instead of equivalence axioms (see Appendix A). The choice for axioms or rules results in equivalent systems, in the sense that the same formulas are provable. However, the choice is not equivalent in a conceptual way.

It seems natural to study subsystems of first-order predicate logic by, for example, dropping the transitivity of the equivalence relation. In the axiomatic setting this can easily be done by leaving out the axiom of transitivity. In the setting without axioms, on the other hand, this clear concept does not seem to coincide with a set of rules.

The other way around, restricting Fitch style flag deduction by leaving out some rules, does not seem to make much sense conceptually.

We will use Fitch style flag deduction with equivalence axioms, which will be indicated by $\mathtt{FD}_{\mathtt{a}}$.

Definition 7.3.1 (Fitch Deductions as Deduction Graphs) *Given a deduction Σ of $\mathtt{FD}_{\mathtt{a}}$ with conclusions B and flags Γ, we define a deduction graph $\hat{\Sigma}$ with top-level-node with label B and free nodes with labels Γ as follows.*

- *Make sure that all subordinate derivations have different eigenvariables, if any. Make also sure that no eigenvariable occurs outside its subordinate derivation. Both can easily be done by renaming eigenvariables.*
- *In all cases where the motivation is not $\forall I$ or $\forall E$, we do the same as in [28]. Make sure that there is only one reference to the line that a $\rightarrow I$ motivation refers to. That can be obtained by inserting an additional line with a Repeat-motivation.*
- *View a formula occurrence B on line n as a node (n, B) and then add edges as follows: if the motivation on line n is $\forall E$,p or $\forall I$,p, and in the $\forall I$-case n is the only line with a reference to p, add an edge from n to p. Now, if there is a subordinate derivation that starts at line i and ends at line j, put a box around nodes $i, \ldots, j$.*

Lemma 7.3.2 *If Σ is a deduction of $\mathtt{FD_a}$ with conclusion B and flags Γ, $\hat{\Sigma}$ is a deduction graph with a top-level node with label B and free nodes with labels Γ.*

7.3.2 Gentzen-Prawitz style natural deduction

Because the language for $\mathtt{DG_U}$s has two kinds of variables, and the language of first-order Gentzen-Prawitz natural deduction has just one, we cannot expect that for every natural deduction proof, we can find a $\mathtt{DG_U}$ with the same conclusion. For instance, there will be no $\mathtt{DG_U}$ with label $(\forall x.\mathtt{P}(x))\rightarrow\mathtt{Q}(x)$ as x occurs both bound and free in this formula. Instead we will define a translation with the following property: Given a natural deduction Σ with conclusion B from assumptions Γ, we define an $\mathtt{SG_U}$ $\bar{\Sigma}$ with a top-level node with label $\bar{B}$ and free nodes with labels $\bar{\Gamma}$, where $\bar{B}$ is B after renaming some bound variables and $\bar{\Gamma}$ is Γ after renaming some bound variables.

Definition 7.3.3 (Natural Deductions as Deduction Graphs) *We define the translation with induction to the structure of the proof.*

Axiom *If φ is an axiom we take (n, φ') where φ' is a renaming of φ such that no variable occurs both bound and free.*

$\rightarrow$-intro *By induction we may assume we have a $\mathtt{DG_U}$ with conclusion (n, ψ') and free nodes $(m_1, \varphi_1), \ldots, (m_k, \varphi_k)$ of renamings of φ. We can find a common renaming φ' of φ, which results in a conclusion (n, ψ''). Now we can, as before, place a box and a box-node $(k, \varphi'\rightarrow\psi'')$.*

$\rightarrow$-elim *By induction we may assume that we have a $\mathtt{DG_U}$ with conclusion (k, φ_1) and one with conclusion $(m, \varphi_2\rightarrow\psi')$. We can rename the labels of the $\mathtt{DG_U}$s to get $\mathtt{DG_U}$s with labels (k, φ') and $(m, \varphi'\rightarrow\psi'')$. Then we add a node (n, ψ'').*

$\forall$-intro *Given a natural deduction with conclusion $\forall x.\varphi$, of which the last step is a $\forall$-introduction on φ. Let Σ be the subproof with conclusion φ. We may assume (by induction) that we have a deduction graph $\bar{\Sigma}$ with top-level node (j, φ'). We now create a box $\mathcal{B}$ consisting of $\bar{\Sigma}$ in which we replace every occurrence of x by a fresh variable, say u, and for all free nodes (p, C) of $\bar{\Sigma}$ we add a new node (p', C) outside $\mathcal{B}$ and an edge from (p, C) to (p', C). Then we place the box-node $(n, \forall x.\varphi')$.*

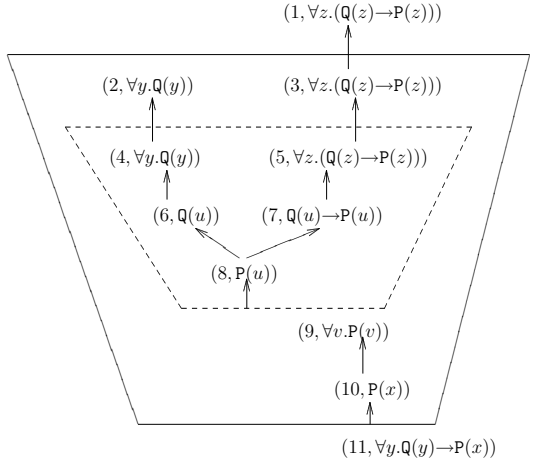
$\forall$-elim *By induction we may assume that we have a $\mathtt{DG_U}$ with conclusion $(n, \forall x.\varphi')$. If the variables of t, the term we want to substitute, occur bound in the $\mathtt{DG_U}$, we rename the labels, resulting in a graph with conclusion $(n, \forall y.\varphi'')$. Then we put a node $(m, \varphi''[t/y])$ with an edge to n.*

Lemma 7.3.4 *If Σ is a natural deduction with conclusion B from assumptions Γ, $\bar{\Sigma}$ is an $\mathtt{SG_U}$ with a top-level-node with label $\bar{B}$ and free nodes with labels $\bar{\Gamma}$, where $\bar{B}$ is B after renaming some bound variables and $\bar{\Gamma}$ is Γ after renaming some bound variables.*

Example 7.3.5 *Consider the following Gentzen-Prawitz style natural deduction proof:*

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall x.\mathtt{Q}(x)]^1}{\mathtt{Q}(x)} \quad \dfrac{\forall x.(\mathtt{Q}(x)\rightarrow\mathtt{P}(x))}{\mathtt{Q}(x)\rightarrow\mathtt{P}(x)}}{\mathtt{P}(x)}}{\forall x.\mathtt{P}(x)}}{\mathtt{P}(x)}\,1}{(\forall x.\mathtt{Q}(x))\rightarrow\mathtt{P}(x)}$$

We can view it as the following $\mathtt{DG_U}$*:*



7.4 Cut-elimination

7.4.1 Transformations

We now also encounter a "detour" in a proof, when a ∀-introduction is immediately followed by a ∀-elimination. Lemma 7.4.3 describes the elimination of a safe ∀-cut.

Not all ∀-cuts are safe, so it might be necessary to apply some transformations to make them safe. These transformations are the same ones as for →-cuts: repeat-elimination, unsharing, and incorporation. The only difference with the transformations on DGs is, that unsharing has become a little more involved, because of the eigenvariable requirement.

Definition 7.4.1 (∀-cut) *A* ∀-cut *in a* $\mathtt{DG_U}$ *G is a subgraph of G consisting of:*

- *A box node* $(n, \forall x.\varphi)$*,*
- *A node* $(p, \varphi[t/x])$*,*
- *A sequence of* R*-nodes* $(s_0, \forall x.\varphi), \ldots, (s_i, \forall x.\varphi)$*,*

- *Edges* $(p, \varphi[t/x]) \longrightarrow\!\triangleright (s_i, \forall x.\varphi) \longrightarrow\!\triangleright \ldots \longrightarrow\!\triangleright (s_0, \forall x.\varphi) \longrightarrow\!\triangleright (n, \forall x.\varphi)$.

We call the node $(n, \forall x.\varphi)$ *the* major premiss *and we call the node* $(p, \varphi[t/x])$ *the* consequence.

Similarly, in a $\rightarrow$-cut, we call $(n, A{\rightarrow}B)$ the major premiss and the node (p, B) the consequence.

Definition 7.4.2 (Safe cut) *Let* $\mathcal{B}$ *be the box associated to box-node* n*. A* $(\forall/{\rightarrow})$*-cut in a* $\mathtt{DG_U}$ G *is* safe *if the following requirements hold:*

- *there is an edge from the consequence to the major premiss and that is the only edge to the major premiss;*
- *the major premiss and the consequence are at the same depth;*

Definition 7.4.3 (Eliminating a safe $\forall$-cut) *The process of* eliminating a safe $\forall$-cut *is the following operation on* $\mathtt{DG_U}$*s (see Figure 7.3):*

- *change the labels* ψ *of the nodes in the box of* n*, to* $\psi[t/u]$*;*
- *remove the box and box-node* $(n, \forall x.\varphi[x/u])$*;*
- *add an edge from* $(p, \varphi[t/x])$ *to* $(j, \varphi[t/u])$ *(the node that* n *pointed to).*

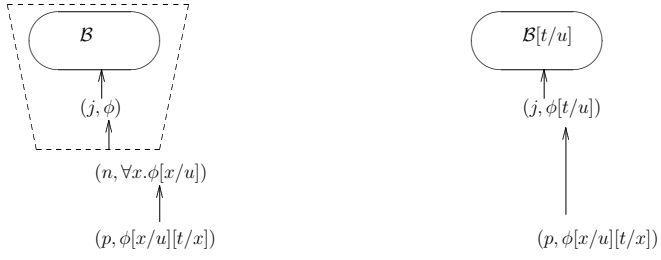


Figure 7.3: Schematic reflection of a safe $\forall$-cut elimination.

Lemma 7.4.4 *If* G *is a* $\mathtt{DG_U}$ *with safe* $\forall$*-cut* c *and* G' *is obtained from* G *by eliminating* c*, then* G' *is also a* $\mathtt{DG_U}$*.*

Proof G' is a closed box directed graph and as box-topological ordering we can take the box-topological ordering of G restricted to the nodes of G'. The node p is an R-node. We have to check that the counterpart in G' of the nodes in the box of n have one of the six types. Let q be a node inside the box in G then:

- Suppose (q, φ) is an A-node in G, then $(q, \varphi[t/u])$ is an A-node in G';

- Suppose (q, ψ) is an $\rightarrow$-E-node in G with edges to $(m, \varphi{\rightarrow}\psi)$ and (p, φ). If both m and p are in $\mathcal{B}$, then in G', $(q, \psi[t/u])$ has edges to $(m, (\varphi{\rightarrow}\psi)[t/u])$ and $(p, \varphi[t/u])$. So in that case q is an $\rightarrow$-E-node. Suppose m is not in $\mathcal{B}$, then u does not occur in it. Hence u does not occur in either p or q, so q is an $\rightarrow$-E-node. Suppose p is not in $\mathcal{B}$, then $(m, (\varphi{\rightarrow}\psi)[t/u]) = (m, \varphi{\rightarrow}(\psi[t/u]))$. So also in this case: q is a $\rightarrow$-E-node in G'.

- Suppose $(q, \varphi{\rightarrow}\psi)$ is an $\rightarrow$-I-node with an outgoing edge to (j, ψ) and such that (r_0, φ), ..., (r_k, φ) are the nodes without outgoing edges in the box. Then in G', q has an edge to $(j, \psi[t/u])$ and $(r_0, \varphi[t/u])$, ..., $(r_k, \varphi[t/u])$ are the nodes without outgoing edges of the box. Because $(\varphi{\rightarrow}\psi)[t/u] = (\varphi[t/u]){\rightarrow}(\psi[t/u])$, q is an $\rightarrow -$ I-node in G'.

- Suppose (q, φ) is an R-node in G with an edge to (l, φ). If l is in G', then $(q, \varphi[t/x])$ has an edge to $(l, \varphi[t/x])$ and is an R-node. If l is not in $\mathcal{B}$, then u does not occur in A, so q is an R-node in G'.

- Suppose $(q, \psi[s/z])$ is an $\forall$-E-node in G with an edge to $(m, \forall z.\psi)$. Then q has in G' label $(\psi[s/z])[t/u]$. This is the same as $(\psi[t/u])([s[t/u]/z])$. If m is not in $\mathcal{B}$, then u does not occur in ψ, so $(\psi[t/u])([s[t/u]/z]) = \psi([s[t/u]/z])$. Hence q is a $\forall$-E-node in G'. If m is in $\mathcal{B}$, then its label is $(\forall z.\psi)[t/u]$, which is the same as $\forall z.(\psi[t/u])$, so also in this case q is a $\forall$-E-node in G'.

- Suppose $(q, \forall z.\psi)$ is an $\forall$-I-node in G with box $\mathcal{B}$ and has an edge to (j, ψ). Remark that $(\forall z.\psi)[t/u] = \forall z.(\psi[t/u])$. So q is an $\forall$-I-node in G'.

Furthermore, no new boxes have been made, so every eigenvariable u still occurs at most once. Suppose v is an eigenvariable of a box $\mathcal{B}'$ of G' and v occurs in the label of a node m outside $\mathcal{B}'$. Then m is in the box $\mathcal{B}$ of G and v is one of the variables occurring in t. But m is in all boxes that p is in, so already in G there is an eigenvariable that occurs in the label of a node outside its box. Contradiction.

We can generalise repeat-elimination, unsharing, and incorporation without much ado. Because after unsharing we still want every eigenvariable to occur just once, this step now includes the renaming of eigenvariables of copied boxes.

Definition 7.4.5 (Cut hidden by repeats) *Let G be a $\mathtt{DG_U}$ with a cut with major premiss (n, φ) and consequence (p, ψ). Suppose G contains a node (n_0, φ), an* R*-node (n_1, φ) and edges $n_1 \longrightarrow\!\triangleright\, n_0$ and $p \longrightarrow\!\triangleright\, n_1$. The* repeat-elimination *at n_0, n_1, p is obtained by:*

- *When an edge points to n_1, redirect it to n_0;*
- *Remove n_1.*

Lemma 7.4.6 *For G a $\mathtt{DG_U}$ with a cut with major premiss (n, φ) and consequence (p, ψ). Suppose G contains a node (n_0, φ), an* R*-node (n_1, φ) and edges $n_1 \longrightarrow\!\triangleright\, n_0$ and $p \longrightarrow\!\triangleright\, n_1$, the repeat-elimination of at n_0, n_1, p is also a $\mathtt{DG_U}$.*

Proof Let G' be the graph after the repeat-elimination step.

1. Box-nodes are always $(\forall/\rightarrow)$-I-nodes. So p is not a box-node. Hence there can be no box-node pointing to n_1 in G. So in G' all box-nodes point to a node in the box (because in G they do). Therefore G' is a closed box directed graph.

2. Take the box-topological ordering of G restricted to the nodes of G'.

3. We have to check that all nodes have one of the six types. Remark that all nodes in G' are of the same type as their counterpart in G.

As the only change with respect to the nodes is, that one has disappeared. Hence the requirement of eigenvariables still hold.

Definition 7.4.7 (Renaming of an eigenvariable) *Let G be a $\mathtt{DG_U}$ with a $\forall$-box $\mathcal{B}$ with eigenvariable u. Let v be a fresh variable. Then the* renaming of u by v *is the graph G in which the labels ψ of the nodes of $\mathcal{B}$ have been replaced by $\psi[v/u]$.*

Lemma 7.4.8 *Let G be a $\mathtt{DG_U}$ with a $\forall$-box $\mathcal{B}$ with eigenvariable u. Let v be a fresh variable. Then the renaming of u by v is a $\mathtt{DG_U}$.*

Proof Let G' be the graph after renaming. As only some labels have changed, the graph is a finite closed box directed graph with a box-topological ordering. We check the type of the box-node and the types of the nodes inside $\mathcal{B}$. Let q be a node inside $\mathcal{B}$ then:

- Suppose (q, φ) is an A-node in G, then $(q, \varphi[v/u])$ is an A-node in G';

- Suppose (q, ψ) is an $\rightarrow$-E-node in G with edges to $(m, \varphi{\rightarrow}\psi)$ and (p, φ). If both m and p are in $\mathcal{B}$, then in G', $(q, \psi[v/u])$ has edges to $(m, (\varphi{\rightarrow}\psi)[v/u])$ and $(p, \varphi[v/u])$. So in that case q is an $\rightarrow$-E-node. Suppose m is not in $\mathcal{B}$, then u does not occur in it. Hence u does not occur in either p or q, so q is an $\rightarrow$-E-node. Suppose p is not in $\mathcal{B}$, then $(m, (\varphi{\rightarrow}\psi)[v/u]) = (m, \varphi{\rightarrow}(\psi[v/u]))$. So also in this case: q is a $\rightarrow$-E-node in G'.

- Suppose $(q, \varphi{\rightarrow}\psi)$ is an $\rightarrow$-I-node with an outgoing edge to (j, ψ) and such that (r_0, φ), ..., (r_k, φ) are the nodes without outgoing edges in the box. Then in G', q has an edge to $(j, \psi[v/u])$ and $(r_0, \varphi[v/u])$, ..., $(r_k, \varphi[v/u])$ are the nodes without outgoing edges of the box. Because $(\varphi{\rightarrow}\psi)[v/u] = (\varphi[v/u]){\rightarrow}(\psi[v/u])$, q is an $\rightarrow$-I-node in G'.

- Suppose (q, φ) is an R-node in G with an edge to (l, φ). If l is in G', then $(q, \varphi[v/u])$ has an edge to $(l, \varphi[v/u])$ and is an R-node. If l is not in $\mathcal{B}$, then u does not occur in A, so q is an R-node in G'.

- Suppose $(q, \psi[t/z])$ is an $\forall$-E-node in G with an edge to $(m, \forall z.\psi)$. Then q has in G' label $(\psi[t/z])[v/u]$. This is the same as $(\psi[v/u])([t[v/u]/z])$. If m is not in $\mathcal{B}$, then u does not occur in ψ, so $(\psi[v/u])([t[v/u]/z]) =$

$\psi([t[v/u]/z])$. Hence q is a $\forall$-E-node in G'. If m is in $\mathcal{B}$, then its label is $(\forall z.\psi)[v/u]$, which is the same as $\forall z.(\psi[v/u])$, so also in this case q is a $\forall$-E-node in G'.

- Suppose $(q, \forall z.\psi)$ is an $\forall$-I-node in G and has an edge to (j, ψ). Because $(\forall z.\psi)[v/u] = \forall z.(\psi[v/u])$, q is an $\forall$-I-node in G'.

Definition 7.4.9 (Cut hidden by sharing) *Let G be a $\mathtt{DG_U}$ with a cut c with major premiss n. Suppose n is a box-node of a box $\mathcal{B}$ and has $k \geq 2$ ingoing edges, from $p_1, \ldots, p_k$. Then the* unsharing of G at nodes $n, p_1, \ldots p_n$ *is obtained by:*

- *making a box $\mathcal{B}'$ that contains a copy of all nodes and edges of $\mathcal{B}$,*
- *copy all outgoing edges of $\mathcal{B}$ to $\mathcal{B}'$ (thus if we had $q \longrightarrow\!\triangleright\, m$ with $q \in \mathcal{B}$, $q' \in \mathcal{B}'$ and $m \notin \mathcal{B}$, then we add $q' \longrightarrow\!\triangleright\, m$, where q' is the copy of $q \in \mathcal{B}'$,*
- *letting $p_2, \ldots, p_k$ point to n' (the box-node of $\mathcal{B}'$) instead of n;*
- *renaming the eigenvariable of $\mathcal{B}'$ and of all boxes contained in $\mathcal{B}'$.*

Lemma 7.4.10 *Let G be a $\mathtt{DG_U}$ with a cut c with major premiss n. Suppose n is a box-node of a box $\mathcal{B}$ and has $k \geq 2$ ingoing edges, from $p_1, \ldots, p_k$. Then the unsharing of G at nodes $n, p_1, \ldots p_n$ is a $\mathtt{DG_U}$.*

Proof Let G' be the graph after unsharing. We make a box-topological ordering of G' by putting all nodes of $\mathcal{B}'$ immediately after the nodes of $\mathcal{B}$ in the box-topological ordering of G. Because of the renaming, the requirement about the eigenvariables holds for G'.

Definition 7.4.11 (Cut hidden by depth-conflict; incorporation) *We have a* depth-conflict *in the $\mathtt{DG_U}$ G, if G contains a cut with major premiss n and consequence p at a greater depth, such that there is an arrow from p to n and that is the only arrow to n. In that case the* incorporation *of G at n, p is obtained by moving $\mathcal{B}_n$, i.e. the box of n, into the box at the lowest depth that includes p but excludes n.*

Lemma 7.4.12 *Suppose G is a $\mathtt{DG_U}$ with a depth conflict. Then the incorporation at the major premiss and the consequence is a $\mathtt{DG_U}$.*

Proof As boxes are used in two different ways, we do a case analysis on the use of the incorporating box.

- Suppose the incorporating box is implicational. Then the types of the nodes and the box-topological ordering remain the same. Because this incorporation does not affect the scopes of the quantifiers, the requirement on eigenvariables hold.

- Suppose the incorporating box $\mathcal{B}$ is a quantifier box. Again, the types of the nodes and the box-topological ordering remain unchanged. Because the eigenvariable of $\mathcal{B}$ and the free variables occurring in labels of nodes of the incorporated box do not coincide, the eigenvariable requirement holds too.

Definition 7.4.13 (Process of $\rightarrow/\forall$-cut-elimination) *Given a* $\mathtt{DG_U}$ G *with a cut* c*, the* process of $\rightarrow/\forall$-cut-elimination *is the following;*

1. *(Repeat elimination) As long as there is no edge from the consequence to the major premiss, perform the appropriate repeat-elimination as described in Definition 7.4.5;*

2. *(Unsharing) If there is an edge from the consequence to the major premiss, but this is not the only edge to the major premiss, perform an appropriate unsharing step, as defined in Definition 7.4.9;*

3. *(Incorporation) As long as the consequence is at a greater depth than the major premiss, perform the appropriate incorporation step, as described in Definition 7.4.11.*

4. *(Eliminating a safe cut) If* c *is safe, perform either the safe* $\rightarrow$*-cut-elimination step, or the safe* $\forall$*-cut-elimination step, as defined in Definition 7.4.3.*

7.4.2 Discussion

We have made some choices in the definition of $\mathtt{DG_U}$s that facilitate the process of cut-elimination. In this section we expound some of the most important choices, and we justify them by showing the difficulties when other choices had prevailed.

The three most important choices are:

1. We deviate from the language of first-order predicate logic and the $\forall$-introduction rule as normally used for Gentzen-Prawitz natural deduction.

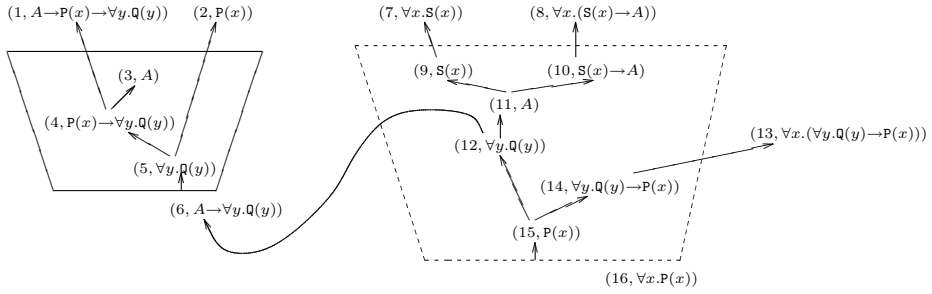
2. We deviate from the side-condition for the $\forall$-introduction rule as normally used in Fitch-style flag deduction. Translating the side-condition for Fitch-style flag deduction to the deduction graph formalism, it reads: The eigenvariable u does not occur in the label of any node that is reachable in one step from $\mathcal{B}$. By requiring that u does not occur in the label of any node that is not in $\mathcal{B}$, we have been much stricter.

3. We require the uniqueness of the eigenvariables.

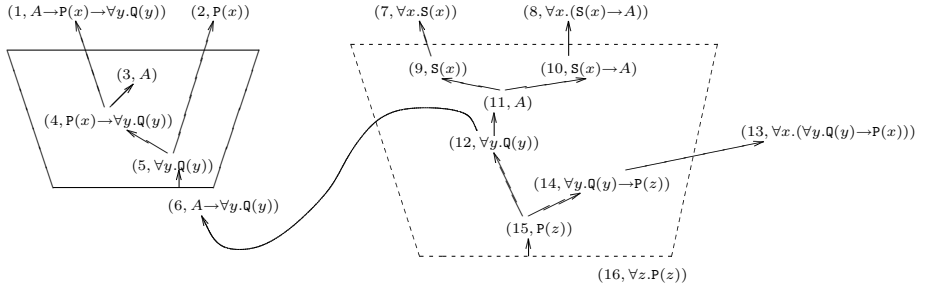
Suppose we would keep to our choices of the language, the $\forall$-introduction rule, and the uniqueness of eigenvariables, but adopt the Fitch-style side-condition for the $\forall$-introduction rule, then this results in having to do an additional renaming in the incorporation step in some cases.

If we would keep to our choice of the language and hence of the ∀-introduction rule, but abandon the requirement of unique eigenvariables and adopt the Fitch-style side-condition, this would move renaming from the unsharing step to the incorporation step.

If we would abandon both or choice for the language, and our side-condition in the ∀-introduction rule, this would cause severe problems in the incorporation step, as the following graph shows.



The eigenvariable of the ∀-box is x. If we would do an incorporation directly, there would be arrows from inside the ∀-box to the nodes 1 and 2 outside the box, in which the variable x occurs free. This is forbidden. We therefore first have to do a renaming of the eigenvariable:



This renaming is not so trivial because it not only involves nodes inside the box but also the x in node 16. But when we rename x in node 16, we also have to rename it in nodes that refer to 16, and propagate that through the graph. This could thus involve any node of G, eventually even nodes 1 and 2. Renaming is hence not just complicated, but it might a priori not even solve the problem.

As this looks like the way we do Gentzen-Prawitz style natural deduction, why doesn't a similar problem –i.e. the necessity to rename variables– occur there? There is no sharing in the example graph, so we can present the deduction quite faithfully as a tree in the following way:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathtt{P}(x)}\quad\dfrac{\dfrac{[A]^1\quad A{\rightarrow}\mathtt{P}(x){\rightarrow}\forall y.\mathtt{Q}(y)}{\mathtt{P}(x){\rightarrow}\forall y.\mathtt{Q}(y)}}{\forall y.\mathtt{Q}(y)}}{A{\rightarrow}\forall y.\mathtt{Q}(y)}\,1\quad\dfrac{\dfrac{\forall x.\mathtt{S}(x)}{\mathtt{S}(x)}\quad\dfrac{\forall x.(\mathtt{S}(x){\rightarrow}A)}{\mathtt{S}(x){\rightarrow}A}}{A}}{\forall y.\mathtt{Q}(y)}\quad\dfrac{\forall x.(\forall y.\mathtt{Q}(y){\rightarrow}\mathtt{P}(x))}{\forall y.\mathtt{Q}(y){\rightarrow}\mathtt{P}(x)}}{\mathtt{P}(x)}}{\forall x.\mathtt{P}(x)}$$

But this is not a correct Gentzen-Prawitz style natural deduction, as the variable x occurs free in some non-discharged assumptions when it gets bound. Apparently the $\forall$-introduction rule in Gentzen-Prawitz style natural deduction is strict enough to prevent the need for renaming variables during $\forall$-cut-elimination.

If we would abandon our choice for the language, but stick to our side-condition and the uniqueness of eigenvariables, we would encounter similar problems.

7.5 Strong Normalisation

To obtain strong normalisation for cut-elimination on $\mathtt{DG_U s}$, we extend the λ-calculus with tupling as defined in Section 4.2.2, and prove strong normalisation for it. Then a reduction preserving translation from $\mathtt{DG_U s}$ to this calculus is defined.

The strong normalisation result we thus get, is again weak: It is assumed that first one cut is made safe and is eliminated, before handling another cut.

Definition 7.5.1 *The* expressions $\mathsf{T}_{\langle\rangle}$ and types of the $\lambda{\rightarrow}\langle\rangle$-calculus for first order predicate logic with universal quantification *are defined as follows:*

1. *For every proposition φ (Definition 7.2.3), variables x^{φ} are expressions of type φ.*
2. *For every expression T of type $\varphi{\rightarrow}\psi$ and every expression S of type φ, (TS) is an expression of type ψ.*
3. *For every expression T of type φ and every variable x^{ψ}, $\lambda x^{\psi}.T$ is an expression of type $\psi{\rightarrow}\varphi$.*
4. *For every expression T of type $\forall x.\varphi$ and every term t (Definition 7.2.1), (Tt) is an expression of type $\varphi[t/x]$.*
5. *For every expression T of type φ, $\lambda y.T[u := y]$ is an expression of type $\forall y.\varphi[y/u]$.*
6. *For every n expressions $T_1, \ldots, T_n$ of types $\varphi_1, \ldots \varphi_n$ respectively, $\langle T_1, \ldots, T_n\rangle$ is an expression of type φ_1.*

Definition 7.5.2 *The reduction rules for the expressions are as follows:*

$$\begin{array}{rcl}
(\lambda x^{\sigma}.M)N & \longrightarrow_{\bar{\beta}} & \langle M, N\rangle \text{ if } x \notin FV(M) \\
(\lambda x^{\sigma}.M)N & \longrightarrow_{\bar{\beta}} & M[x := N] \text{ if } x \in FV(M) \\
(\lambda y.M)t & \longrightarrow_{\bar{\beta}} & M[y := t] \\
\langle M, P_1, \ldots, P_k\rangle N & \longrightarrow_{\bar{\beta}} & \langle MN, P_1, \ldots, P_k\rangle \\
\langle M, P_1, \ldots, P_k\rangle t & \longrightarrow_{\bar{\beta}} & \langle Mt, P_1, \ldots, P_k\rangle \\
N\langle M, P_1, \ldots, P_k\rangle & \longrightarrow_{\bar{\beta}} & \langle NM, P_1, \ldots P_k\rangle \\
\langle \ldots, \langle M, P_1, \ldots, P_k\rangle, \ldots\rangle & \longrightarrow_{\bar{\beta}} & \langle \ldots, M, P_1, \ldots, P_k, \ldots\rangle
\end{array}$$

As can be observed from the typing and the reduction rules, the $N_1, \ldots, N_k$ in $\langle M, N_1, \ldots, N_k\rangle$ act as a kind of 'garbage'. The order of the terms in $N_1, \ldots, N_k$ is irrelevant and we therefore consider terms *modulo* permutation of these vectors, which we will write as $\equiv_p$.

Definition 7.5.3 *Given a deduction graph G and a node n in G, we define the λ-term $(\!|G, n|\!)$ as follows (by induction on the number of nodes of G).*

- A *If (n, A) has no outgoing edges, $(\!|G, n|\!) := x_n^A$,*
- $\to\mathsf{E}$ *If $(n, B) \twoheadrightarrow (m, A{\to}B)$, and $(n, B) \twoheadrightarrow (p, A)$, define $(\!|G, n|\!) := (\!|G, m|\!)\,(\!|G, p|\!)$.*
- R *If $(n, A) \longrightarrow\!\!\triangleright\, (m, A)$, define $(\!|G, n|\!) := (\!|G, m|\!)$*
- $\to\mathsf{I}$ *If $(n, A{\to}B)$ is a box-node with $(n, A{\to}B) \longrightarrow\!\!\triangleright\, (j, B)$, the free nodes of the box are $n_1, \ldots, n_k$ and the nodes without incoming edges inside the box are $m_1, \ldots, m_p$, then*

$$(\!|G, n|\!) := \lambda x^A.\langle (\!|G, j|\!), (\!|G, m_1|\!), \ldots, (\!|G, m_p|\!)\rangle[x_{n_1} := x, \ldots, x_{n_k} := x].$$

- $\forall\mathsf{E}$ *If $(n, \varphi[t/y]) \longrightarrow\!\!\triangleright\, (p, \forall y.\varphi)$, define $(\!|G, n|\!) := (\!|G, p|\!)t$.*
- $\forall\mathsf{I}$ *If $(n, \forall y.\varphi[y/u]) \mathrel{--}\!\!\triangleright\, (j, \varphi)$ and the nodes without incoming edges are $m_1, \ldots, m_p$, then*

$$(\!|G, n|\!) := \lambda y.\langle (\!|G, j|\!), (\!|G, m_1|\!), \ldots, (\!|G, m_p|\!)\rangle[u := y]$$

The interpretation of the deduction graph G, $(\!|G|\!)$, is defined as $\langle (\!|G, r_1|\!), \ldots, (\!|G, r_l|\!)\rangle$, where $r_1, \ldots, r_l$ are the top-level nodes without incoming edges in the deduction graph G.

Definition 7.5.4 *A* $\lambda{\to}\langle\rangle$ context *is given by the following abstract syntax $K[-]$.*

$$K[-] := [-] \mid \mathsf{T}_{\langle\rangle}K[-] \mid K[-]\mathsf{T}_{\langle\rangle}$$

So a $\lambda{\to}\langle\rangle$ *context* is a $\lambda{\to}\langle\rangle$-term consisting only of applications (no abstractions) with one open place. The following is immediate by induction on $K[-]$.

Lemma 7.5.5 *For all $\lambda{\to}\langle\rangle$ contexts $K[-]$ and $\lambda{\to}\langle\rangle$-terms $M, N_1, \ldots, N_k$*

$$K[\langle M, N_1, \ldots, N_k\rangle] \twoheadrightarrow_{\bar\beta} \langle K[M], N_1, \ldots, N_k\rangle.$$

Lemma 7.5.6 ($\forall$ Cut-elimination is $\bar\beta$-reduction in $\lambda{\to}\langle\rangle$)
If G' is obtained from G by a $\forall$-cut-elimination, then $(\!|G|\!) \twoheadrightarrow^{+}_{\bar\beta} (\!|G'|\!)$.

Proof By induction on the structure of G. Note that, if G' is obtained from G via a repeat-elimination step, an unsharing step or an incorporation step, then $(\!|G|\!) \equiv_p (\!|G'|\!)$. If G' is obtained from G by contracting a safe cut, say with $(p, \varphi[t/y])$ being the consequence, then $(\!|G, p|\!)$ is a subterm of $(\!|G|\!)$, possibly several times. We look at the subterm $(\!|G, p|\!)$ and we observe the following:

$$\begin{array}{rcl}
(\!|G, p|\!) & \equiv & (\!|G, n|\!)t \\
& \equiv & (\lambda y.\langle (\!|G, j|\!), (\!|G, m_1|\!), \ldots, (\!|G, m_r|\!)\rangle[u := y])t \\
& \to_{\bar\beta} & \langle (\!|G, j|\!), (\!|G, m_1|\!), \ldots, (\!|G, m_r|\!)\rangle[u := y][y := t] \\
& \equiv & \langle (\!|G', p|\!), (\!|G, m_1|\!)^*, \ldots, (\!|G, m_r|\!)^*\rangle
\end{array}$$

where the superscript * denotes the substitution $[u := t]$. If we look at $(\!|G, p|\!)$ as a subterm of $(\!|G|\!)$, we first note that it occurs as a subterm of a $(\!|\ldots, \ldots|\!)$ expression as follows: $(\!|G|\!) = (\!|\ldots, K[(\!|G, p|\!)], \ldots|\!)$. In a cut-elimination, the box has been removed and the nodes $m_1, \ldots, m_r$ have moved one level up (in terms of depth), thus

$$\begin{array}{rcl}
(\!|G|\!) & \equiv & \langle \ldots, K[(\!|G, p|\!)], \ldots\rangle \\
& \twoheadrightarrow^{+}_{\bar\beta} & \langle \ldots, K[\langle (\!|G', p|\!), (\!|G, m_1|\!)^*, \ldots, (\!|G, m_r|\!)^*\rangle], \ldots\rangle \\
& \twoheadrightarrow_{\bar\beta} & \langle \ldots, \langle K[(\!|G', p|\!)], (\!|G, m_1|\!)^*, \ldots, (\!|G, m_r|\!)^*\rangle, \ldots\rangle \\
& \to_{\bar\beta} & \langle \ldots, K[(\!|G', p|\!)], (\!|G, m_1|\!)^*, \ldots, (\!|G, m_r|\!)^*, \ldots\rangle \\
& \equiv_p & (\!|G'|\!)
\end{array}$$

Theorem 7.5.7 *The process of cut-elimination is terminating for* $\mathtt{DG_U}$*s.*

Proof Analogous to the DG case.

7.6 Connection with Proof Nets

In [29] we have seen a correspondence between a variant of DGs and proof nets of MELL. We remarked that there are some superficial similarities between the two: both have boxes and both enable sharing (contraction). Using this, we were able to define a translation from these deduction graphs to proof nets that preserves reduction.

Proof nets seem also in the way they handle quantification fairly close to deduction graphs. In the early days [36] boxes were used do delimit the scope of a quantification. Later (see for example [86]), this was put aside and replaced

by global correctness criteria. It seems plausible that in deduction graphs too we could omit boxes for this use. We have not done this as deduction graphs serve another purpose than proof nets, and leaving out the ∀-boxes would make the deduction graphs less perspicuous. This discrepancy in the handling of quantification does not seem to jeopardise the aim to extend the translation given in [29]: Because $(\forall x.\varphi)^* = !(\forall x.\varphi^*)$ (where $()^*$ is Girard's translation), it is the *exponential* box that should act like the ∀-boxes in deduction graphs anyway.

The main difficulty in both proof nets and deduction graphs is that during cut-elimination it is in some cases necessary to do a renaming. In anticipation to this, we have changed the ∀-introduction rule for deduction graphs and we have used two kinds of variables: one kind for bound uses, and one for free uses (see also [85]).

In [37], Girard discusses proof nets of MLL with quantifiers. Note that, as these proof nets do not include the exponential rules, this results in a simpler system. His approach is similar to ours. He replaces some free variables by constants, which reminds of our solution with two kinds of variables. He also insists on uniqueness of the eigenvariable. About renaming he says:

> In practice, it would be crazy to rename bound variables (...).

Luckily, as there is no copying going on in the cut-elimination of MLL, renaming is nowhere necessary.

This changes when we shift our attention to MELL proof nets with quantification. The most complete study of this can be found in [86], and although it handles only second order quantification explicitly, it is generally assumed [36] [86], that first order quantifiers do not provide additional difficulties.

Here another approach has been taken. Instead of discriminating between different kinds of variables, an equivalence relation on the formulas is defined, making two formulas the same when one can be obtained from the other by renaming bound variables. Deviating from [86], this line might be pursued as follows: [1]

1. Define formulas;
2. Define proof-structures;
3. Define the equivalence relation on formulas;
4. Extend the equivalence relation on proof-structures.

Once this has been done, it needs to be shown that after cut-elimination on a proof-net, one gets a proof-structure that is equivalent to a proof net.

This plan has two difficulties, the first being the exact definition of equivalence relation on proof structures. Just saying that two proof structures are equivalent, when they have the same structure and when formulas at the same places are equivalent, would not suffice. In addition, it should also consider

[1] Personal communication Lorenzo Tortora de Falco

renaming of *free* variables in formulas that will get bound somewhere else in the structure.

Secondly, it could be rather complicated to find an equivalent proof-net after cut-elimination. This problem is very similar to the ones discussed in Section 7.4.2. It is not at all clear how this renaming can be done for example after c-b-reduction (copying a box).

Another way out would be to extend the idea used in [37], similarly to deduction graphs: Change the $\forall$-rule and work with two kinds of variables. This might very well work.

Whence proof nets with quantifiers are defined properly and completely, it seems likely that we can define a reduction-preserving translation from a variant of deduction graphs with universal quantification to them.

Part II

Formalised Mathematics on the Web

Chapter 8

Formalised Interactive Algebra Course Notes

8.1 Formalising Mathematics

8.1.1 Aims of Formalising Mathematics

Mathematics is probably the only science that claims to produce ever-lasting truths.

In earlier times people sometimes found "exceptions" to a mathematical result, without dismissing the result altogether. Cauchy, for instance, stated that the sum of a sequence of continuous functions was again continuous. When Abel found a sequence of continuous functions that did not act as stated by Cauchy's theorem in the beginning of the 19th century, he called it an exception.

The light way in which people handled what we would now call counterexamples, might be explained by their emphasis on the usefulness of the theorems, instead of on the correctness [75]. Nowadays, however, we only consider something a theorem, when it holds in all cases that could possibly arise.

This claim of persistence does not take into account that mathematics is a human activity, and that humans make mistakes. So, in reality, when a statement has been proven, we consider it valid only as long as no errors have been found. As time passes and many mathematicians have studied the proof into detail without finding errors, the assumption that the proof is correct and hence that the statement holds, becomes more and more likely.

In spite of the impossibility to be totally sure of any mathematical result, it does not mean that we should reject mathematics altogether. Rather it means that a mathematician should strive for the highest degree of correctness attainable, as to diminish the chances that a proof contains any errors.

One thing that humans are fairly bad at, and computers can do quite well, is bookkeeping. A possible way to obtain a high degree of correctness is thus to use a computer to do the bookkeeping, for instance to provide help in the

development of a proof by recording all cases that have to be handled and presenting them one by one to the user, and to check that the provided proof is correct. This, of course, only makes sense, if we trust the computer-program that does the bookkeeping, the so-called *proof-assistant*. (These days most proof-assistants do more than just bookkeeping: they also prove technically simple problems by themselves.)

In order to open the door for computer-assisted proofs, a proof should be done in a format that the computer can handle, that is: the proof should be formal. The process of making a proof or definition inside a proof-assistant is hence called *formalising mathematics*. After a formal proof has been obtained, the fact that it is formal can be used for other applications, e.g. to automatically extract a program out of it [23], or to easily find the axioms a proof uses.

There is also a disadvantage to the use of formalised mathematics: formal proofs have the tendency to be harder to read than the informal ones. This has the following consequences:

1. It can be hard to say whether some formal proof indeed proves some informal statement, if that makes any sense at all.

2. The chances that someone finds a mistake by reading a formal proof become quite small.

3. It is questionable whether a formal proof can be used to teach someone mathematics, that is: to clarify the line of mathematical thought.

At the moment there are many proof-assistants in use. PVS [73], NuPRL [67], Mizar [63], Coq [19], Agda [1], Isabelle [43], and HOL [40] are some of them.

8.1.2 Propositions as Types

Some proof assistants, like Coq and Agda, are based on type theory. This is possible thanks to the *Curry-Howard-de Bruijn isomorphism*. That is the idea that propositions can be seen as types, and the proof of a proposition can be seen as a λ-term of the associated type. In this way, checking whether a proof proves a certain proposition amounts to checking whether a term has a certain type.

Type theory and λ-calculus are especially suited for computations. According to the *Poincaré principle* [10] computation itself does not require any proof. This principle is made explicit by the *axiom scheme of conversion*.

In higher order logic a simple instance of this scheme can sometimes be found ([10]):

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \psi} \text{ if } \varphi =_\beta \psi$$

expressing that we may directly conclude a proposition from one to which it is β-equal.

In proof-assistants based on type-theory, the equality in the side-condition is often extended with for example ι-equality (for recursive definitions) and δ-equality (unfolding and folding of definitions), leading to the following scheme:

$$\frac{M{:}\varphi \quad \psi{:}\text{Prop}}{M{:}\psi} \text{ if } \varphi =_{\beta\iota\delta} \psi$$

This states that a proof of a proposition φ may be taken as a proof of any other proposition that is $\beta\iota\delta$-equal to it.

The use of the Poincaré principle can shorten a derivation dramatically.

8.1.3 Coq and C-CoRN

Coq

Coq [19] is a proof-assistant based on the type theory called *Calculus of (Co)Inductive Constructions* (CIC) for constructive mathematics developed at Inria. It separates the process of formalising into two independent phases: that of *proof-construction*, and that of *proof-checking*.

In the construction phase, the user gives commands to Coq, the so-called *tactics*, which result in the construction of a λ-term, the *proof-term*. After the execution of an applicable tactic, Coq shows to the user what still has to be proven. When nothing has to be proven anymore according to the part of Coq that is in charge of the construction, the first phase is over. The sequence of given tactics is called the *proof-script* (See Figure 8.1).

```
Lemma cs_unique_unit :
forall (S:CSemiGroup) (e f:S),
                          (is_unit e) /\ (is_unit f) -> e[=]f.

Proof.
 intros S e f.       intros H2 H3.
 unfold is_unit.     elim (H1 e).
 intros H.           clear H1.
 elim H.             intros H4 H5.
 clear H.            astepr (e[+]f).
 intros H0 H1.       astepl (e[+]f).
 elim (H0 f).        apply eq_reflexive.
 clear H0.           Qed.
```

Figure 8.1: Proof-script for the proof of uniqueness of units in semi-groups.

Then the proof-checking phase starts. This consists of inspecting the proof-term constructed in the previous phase, in order to check that it corresponds to a proof with the desired conclusion. (This bipartition into a proof-construction phase and a proof-checking phase is a simplification, though: Proof-checking is already used during proof-construction.)

If we want to know whether some statement is true, it is enough to know that there exists a proof with that statement as conclusion, i.e. there exists a λ-term of which the given statement is the type. The way this term came into being is of no importance. (This does not mean that all ways of constructing a proof-term for a statement, results in the same proof-term.)

This leads to the important remark that to trust the proofs of Coq, we only have to trust the proof-checker, a part of the program that is relatively small. Because of its limited size, the chances that the programmer has made mistakes in this division is comparatively small. Moreover, we should in principle be able to check it ourselves.

A proof-assistant like Coq that has a small proof-checker is said to satisfy the *de-Bruijn-criterion* [10].

Besides, several attempts have been made to make a readable proof-script for Coq [8], [88], [34]. This should not only meet some of the consequences of the lack of readability raised in the previous section, but it should foremost facilitate the actual construction of the proofs.

Users can add *automation* to Coq: Algorithms with which Coq can prove statements by itself. A method that is very often used for this is *reflexion*. In short, reflexion exploits the possibility to accept a statement on purely syntactical grounds by mere computation applying a generic lemma, and in Coq computation does not require a proof. Of course, parts of a proof that Coq comes up with itself, including the ones by reflexion, also build up parts of the proof-term.

C-CoRN

When a mathematician writes a mathematical document, be it an article, a book, or even course notes, he very rarely starts *ab ovo*. Not the axioms serve as starting point, but rather all results that have been obtained over the last centuries. Starting from the very beginning would be highly inefficient, and it would distract from the actual subject of the document.

The same holds for formalised mathematics. When formalising a proof, say, about real functions, it would be inconvenient to first have to formalise basic results about natural numbers. It is therefore of great importance for every proof-assistant, that there exists a *library* for it, that is: a collection of (basic) mathematics already formalised in that proof assistant, that can be reused.

C-CoRN [20] [22], the Constructive Coq Repository at Nijmegen, is such a library. It started of as the FTA-project [33], which had the goal of formalising the fundamental theorem of algebra (FTA). Working towards this, great care has been taken to make the intermediate results *reusable*. So, lemmas necessary for the FTA were formulated in a very general way, hoping that they would not only be of use in this single project, but that they would also serve many more projects to come.

C-CoRN has grown out over the years to a library of 4 MB of proof-scripts and necessary tools. It includes algebra like an algebraic hierarchy [32], consisting of semi-groups, monoids, groups, rings, fields, ordered fields and metric

spaces, and the fundamental theorem of algebra. There is also a large amount of analysis, including real numbers, complex numbers, polynomials, real valued functions, differentiation/ integration, the intermediate value theorem, Taylor's theorem, and the fundamental theorem of calculus [21].

8.1.4 Helm

Helm's Philosophy

Whence a library has been realised, a new problem arises: For how, in all these files of tactic scripts, should a user find the theorems he needs? A *documentation tool*, like the one distributed with Coq, CoqDoc [19], takes the proof-script, tries to make it a bit more readable by replacing some of the code by mathematical notation, and puts it on the web.

Helm [39] [7] [5], the Hypertextual Electronic Library of Mathematics, developed at the University of Bologna, chooses for another approach. On the one hand, it does output HTML (HyperText Markup Language [42]), just like CoqDoc. This is an obvious choice, because it enables the user to utilise Helm without installing any extra programs, as all browsers support HTML. (Besides, Helm also outputs MathML [60].) On the other hand, it does not take the *proof-script* as starting point, but, instead, the *proof-term*. This may seem remarkable, as the proof-term is even less readable than the proof-script. Moreover, it seems that the construction of the proof, which is not reflected in the proof-term, contains valuable information about the line of thought. Still, there are good reasons to start from the proof-term, the most important being *stability* and *sustainability*.

Coq and most other proof-assistants are not finished programs. Many people are still working on them: Making them easier to use and exploring theoretical extensions. This means that these programs change now and then, and these changes often affect the tactic language too. When a new version of a proof-assistant is released, all proofs of the library should be redone. (Often this can be done automatically.) The point is: the proof-script of a library is unstable. This does not mean that the proof-terms it constructs change too. (And if they do, the change seldom involves the syntax or semantics of the proof-term language.) As Helm aims for *stability*, the proof-terms seem a better candidate.

Furthermore, both the completeness of information and the stability of the proof-terms, and the universal agreement on its semantics, make these terms in principle suitable for long-term preservation of the information: Coq might no longer be used at some point, but the proof-terms are convincing by themselves. This does not hold for the proof-script. This makes Helm *sustainable*.

Another feature of Helm is that the whole process, so from proof-terms to readable proofs, is done by XSLT (Extensible Stylesheet Language Transformations [90]), translating between different varieties of XML (Extensible Markup Language [89]). Again, this choice might be questionable. For one, as XSLT is very slow, Helm is quite slow as well. Furthermore, it is one of the reasons that the amount of programming code is big, and that the code is not very ac-

cessible. The latter is of no concern to the user, but a programmer might need a considerable time of study to change just something small in the program. However, also here there were good reasons for this choice.

Firstly, XML-languages are by their openness very useful to store and exchange information. Using XML for Helm opens the possibility for Helm to be more than a documentation tool. XML also facilitate *interoperability*, that is: the exchange of information between various programs; in this case for example between proof assistants and computer algebra systems.

Secondly, using XML contributes also to the sustainability of the program.

Whence chosen for XML, the choice for XSLT follows almost immediately, because XSLT has been specially designed for the transformation of XML-languages. Moreover, the limited expressive power of XSLT has the philosophical advantage that it forces rendering to be a simple operation, not involving major transformations on the proofs. If it would be very sophisticated, one might loose confidence in what has been actually proven in the proof-assistant.

Helm in Practice

The process of transforming a Coq proof to an HTML or MathML-Presentation proof is done as follows. From the Coq files, the lambda-term is exported to CICML, a for this purpose designed XML-language. The XML exportation module is currently a standard component of the Coq distribution.

The raw XML encoding used by Helm to store the information in the repository is transformed into a presentational markup by a suitable set of stylesheets. This transformation process is split, for modularity reasons, in two main parts, passing through an intermediate "content" markup, which is provided by MathML-Content and OMDoc (Open Mathematical Document [68]). The idea is that different foundational dialects, like Coq and NuPRL, may be mapped to a same content description, and on the other side, the same content can be translated to different presentational markups. (See Figure 8.2.)

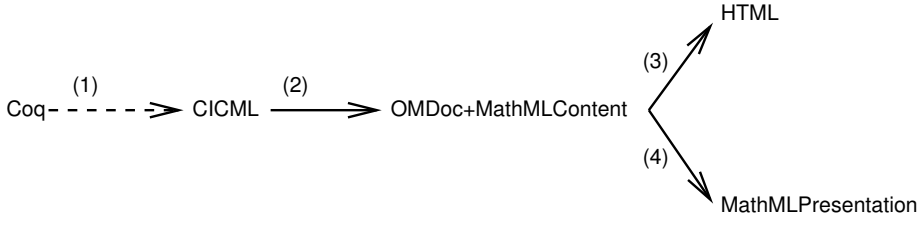


Figure 8.2: The transformation process of Helm.

The exportation (1) is done by the exportation module [77]. The transformations (2), (3) and (4) are done by XSLT stylesheets of Helm. These transformations are done *dynamically*; that means that Helm only transforms and renders a definition or lemma on request, so in a lazy way.

The stylesheets that handle the structure of the proofs are predefined in

Helm. This is possible, because Coq only knows a restricted set of rules. Mathematical objects, on the other hand, can be freely added by the user, and can therefore not be prepared in advance. Because it cannot be expected that a user writes the XSLT-code for the transformations himself, Helm provides a way to automatically generate them.

To automatically generate stylesheets, Helm offers a simple XML language for the user to describe notation for the formally defined mathematical objects, and it offers a set of transformations in XSLT, the so-called *meta-stylesheets*, which translate the description in the simple XML language to stylesheets in charge of the notational part of the transformations (2), (3) and (4) (see [54] for the first implementation). Moreover, links are automatically added from one level to the generating expression at the previous level. These links are necessary, because the transformation is done on the fly, and hence the system needs to track the request of the user on the lowest level.

Describing the notation (in the first implementation) for the "less than" operator in the simplified language simply amounts in giving (Figure 8.3):

1. its name;
2. its URI (unique identifiers) of the operator at the CICML level;
3. its arity;
4. the name of the MathML Content element that it must be mapped to;
5. the name of the MathMLPresentation element and HTML element/Unicode symbol that it must be mapped to for presentation.

This information can easily be provided by the user by directly editing the XML file. The figure also shows how names are associated to MathMLPresentation elements and HTML elements.

Below we give an example of a proof in C-CoRN, see Figure 8.1, as rendered by the Helm tools (Figure 8.4). Observe that the C-CoRN proof is much more verbose than the one that can be found in text books, for example in [18], which just reads

$$e = e + f = f.$$

On the other side, the C-CoRN proof-script (Figure 8.1) is less readable than this, and the Helm rendering is much more structured than the proof-term which it stems from. Some of the details are "hidden" under links (green, when seen in color), which are automatically added by Helm: clicking these links unfolds the details. In this example, these links are the crossed box before "Assumptions", which unfolds explicitly the local assumptions of the lemma, and the "Proof of", which gives an explicit proof of this fact. Words and symbols that originate from defined notions in Coq are links to their definition (blue). So, "CSemiGroup" links to the definition of constructive semi-groups in C-CoRN, as rendered by Helm. Similarly, "is a unit of" refers to the mathematical definition of being a unit of some semi-group. Only the predefined notions of Coq (blue), like "∀", and natural language explanation of the proof steps (red), like "we must prove" and "or equivalently", are no links.

```
<Symbols type="MathMLPresentation">
   <symbol tag="lt" value="&lt;" priority="40" />
</Symbols>

<Symbols type="HTML">
   <symbol tag="lt" priority="40" >
      <case type="symbol" value="&#60;"/>
      <case type="unicode" value="&#60;&#32;"/>
   </symbol>
</Symbols>

<Operator
name = "LESS THAN"
uri = "cic:/Coq/Init/Peano/lt.con | cic:/Coq/ZArith/BinInt/Zlt.con"
arity = "2"
c-tag = "lt"
p-tag = "lt"
h-tag = "lt"/>
```

Figure 8.3: Rendering information for "less than", supplied by the user.

cic:/CoRN/algebra/CSemiGroups/cs_unique_unit.con [search]

```
DEFINITION cs_unique_unit()
TYPE =
    ∀S:CSemiGroup.∀e:S.∀f:S.e is a unit of S∧f is a unit of S→e=f
BODY =
    ⊞Assumptions
      we must prove e is a unit of S∧f is a unit of S→e=f
      or equivalently (∀a:S.e+a=a∧a+e=a)∧(∀a:S.f+a=a∧a+f=a)→e=f
      suppose H: (∀a:S.e+a=a∧a+e=a)∧(∀a:S.f+a=a∧a+f=a)
        consider H
        we have:
        (H0) ∀a:S.e+a=a∧a+e=a
        (H1) ∀a:S.f+a=a∧a+f=a
        by (H0 .)
        we have:
        (H2) e+f=f
        (H3) f+e=f
        by (H1 .)
        we have:
        (H4) f+e=e
        (H5) e+f=e
        we have the following chain of (in-)equalities:
         e
            by
            Proof of e=e+f
          = e+f
            by H2
          = f
        we proved e=f
      we proved (∀a:S.e+a=a∧a+e=a)∧(∀a:S.f+a=a∧a+f=a)→e=f
      that is equivalent to e is a unit of S∧f is a unit of S→e=f
    we proved ∀S:CSemiGroup.∀e:S.∀f:S.e is a unit of S∧f is a unit of S→e=f
```

Figure 8.4: Rendering of a formal proof by Helm.

8.2 Course Notes with Formalised Proofs and Definitions

8.2.1 Introduction

One of the aims of the European IST Project MoWGLI [65] was the development of a suitable technology supporting the creation of web-based didactic material out of repositories of formal mathematical knowledge. As a test case we actually created web-based course notes with formal mathematics of C-CoRN in it, using Helm.

A course is much more than a set of definitions and theorems. It has been divided in chapters and paragraphs, and it may sometimes sacrifice formality to favour intuitions and explanation. Furthermore, it contains plenty examples and exercises.

Thus, to develop course notes from C-CoRN, it is not a reasonable idea to start from the proof-script and integrate it with exercises and explanations. So, in the MoWGLI project a different approach has been taken: Starting from pre-existing course notes, IDA [18] (see Section 8.2.2), seen as hypertextual document, formal definitions, lemmas and examples have been embedded.

The work that had to be done concretely to create this web-based didactic material, can be roughly divided into four tasks:

1. Formalising the mathematics covered in some pages of IDA, as far as they were not already present in C-CoRN.

2. Extending and improving the possibilities to automatically generate stylesheets in Helm, to such extent that no notational stylesheet had to be done by hand anymore. Then, providing the actual notation for the mathematical objects in the formal mathematics meant for IDA.

3. Adapting Helm in order to be able to incorporate pages of IDA, without loss of IDA's already existing interactive functionality.

4. Adapting Helm in order to allow control over the way that formal mathematical objects would be rendered: This was needed in order to not only allow the incorporated IDA to show the lemmas and definitions of C-CoRN instead of the informal ones, but also to let mathematical objects in the textual flow refer to formal objects in the library.

The formalisation of the mathematics will be discussed in Section 8.3; Section 8.4 treats the adaptations of the possibilities for the automatic generation of notational stylesheets. As items 3 and 4 have mainly been done by Lionel Elie Mamane, they will not further be discussed here. In Section 8.5 we discuss the created formalised course notes, and in Section 8.6 we come back to the question whether the MoWGLI-project has succeeded in developing the right technology for this.

8.2.2 IDA

IDA [18] is an interactive course on algebra, which has been developed at the Eindhoven University of Technology (NL), for first year mathematics students. The IDA course notes have been developed over the years in the context of a first year class on algebra. It consists of a book with a CD-ROM. (Before the book + CD-ROM were published by Springer, β-versions of the material were available through the web.) The material on the CD-ROM is the same as in the book, but there is additional functionality:

- Alternative ways of browsing through the book.
- Some hyperlinks to definitions and lemmas.
- Applets that show algorithms (e.g. Euclid's algorithm).
- Multiple choice exercises.

IDA does not provide a formal treatment of proofs. Proofs in IDA are like proofs in ordinary mathematical text books: plain text.

8.3 Formalising IDA

8.3.1 Introduction

Although C-CoRN contains much algebra, only about 20% of the formal mathematics we have used directly in the document, was present at the time we started. This low percentage is partly due to the fact that C-CoRN did not contain many examples, which can be explained by its origin (see Section 8.1.3). It must be said that many lemmas that were not contained in C-CoRN, were fairly easy to formalise using the theory that C-CoRN did contain. However, the formalising task brought about two problems:

1. IDA contained a non-constructive theorem;
2. Whence formalised, it appeared that Helm could not render some lemmas in a satisfactory way.

In the following paragraphs we discuss how we have decided to handle the non-constructive mathematics. Furthermore, we explain why Helm could not render some lemmas, and we offer a method to avoid this problem.

8.3.2 Handling non-constructive proofs

C-CoRN is a library of constructive mathematics, and IDA is a course in classical algebra, although it has a special focus on algorithms. Because the structures in C-CoRN are all equipped with an apartness relation, it meant that we had to prove more than has been proven in IDA. Instead of showing, for instance, that some set with an operator and an equality was a semi-group, we had to

define also an apartness relation on this set and show that it was a constructive semi-group. So we had to do a bit more work, but in general it did not trouble us much.

Only in one place the disagreement of IDA and C-CoRN concerning the logic was problematic: the characterisation of cyclic monoids.

Definition 8.3.1 *A monoid that can be generated by a single element is called* cyclic.

One would expect from the name "*cyclic* monoid" to find some cyclicality in the generation of the elements of such a monoid, but this is not necessarily the case. The monoid $(\mathbb{N}, +, 0)$, for example, is also a cyclic monoid: it is generated by 1.

The Characterisation of Cyclic Monoids tells us, that these are exactly the two cases that could occur: a cyclic monoid either has a cycle, or it is isomorphic to $(\mathbb{N}, +, 0)$.

To state this theorem, the idea of a monoid with a cycle is made precise. For every $k, n \in \mathbb{N}$, $n > 0$ a cyclic monoid $C_{k,n}$ is defined. The carrier of this monoid is the set $\{e, c, \ldots, c^{k+n-1}\}$ and the operation is defined as follows:

- $c^j * c^i = c^{j+i}$, if $j + i < k + n$;
- $c^j * c^i = c^{k+((j+i-k) \bmod n)}$, if $j + i > k + n - 1$;
- c^0 is the unit e.

Figure 8.5 shows a visualisation of the orbit of e under $*c$.

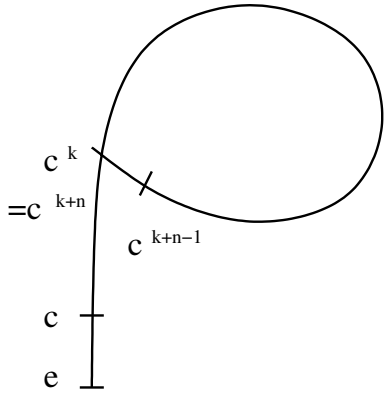


Figure 8.5: The orbit of e

Theorem 8.3.2 (Characterisation of cyclic monoids.) *Every cyclic monoid is isomorphic with either $C_{k,l}$ for certain $k, l \in \mathbb{N}$ or $(\mathbb{N}, +, 0)$.*

However, this bipartition fails in constructive mathematics. We will show this by a Brouwerian counterexample.

Definition 8.3.3 *We introduce the following abbreviations:*

$n < k_{99}$	*In the first n digits of the decimal expansion of π, there is no block of 99 successive nines.*
$n = k_{99}$	*The nth digit of the decimal expansion of π closes for the first time a block of 99 successive nines.*
$n > k_{99}$	*There exists a block of 99 successive nines in the first $n-1$ digits of the decimal expansion of π.*

Similarly the abbreviation $n \leq k_{99}$ and $n \geq k_{99}$ can now be introduced.

Counterexample 1
Define $M := \{n | n \in \mathbb{N} | n < k_{99}\}$. Define the operation $*_M$ by:

$$i *_M j = \begin{cases} i + j & \text{if } i + j < k_{99} \\ i + j \text{ mod } n, \text{ where } n = k_{99} & \text{if } i + j \geq k_{99} \end{cases}$$

Then it is clear that $(M, *_M, 0)$ is a monoid. But saying that $(M, *_M, 0)$ is isomorphic with either $C_{k,l}$ for certain $k, l \in \mathbb{N}$ or $(\mathbb{N}, +, 0)$ implies that the decimal expansion of π contains a block of 99 nines or it doesn't, which is audacious.

Because we could thus not have any hope to prove this theorem in Coq in the form it was presented in IDA, we have adapted the theorem such that it could be proven therein. To understand our adaptation, it is explanatory to study the structure of the original proof.

Proof Suppose that $(C, *, e)$ is a cyclic monoid generated by the element c of C. We make the following case distinction.

- **There are $k < l$ with $c^k = c^l$**
  Let k and l be the smallest pair (in lexicographical order) with this property.
  $\vdots$
  ⟨construction of a map from $C_{k,n}$ for certain $k, n \in \mathbb{N}$ to C and a proof that it is an isomorphism⟩
  $\vdots$
  Hence C is isomorphic with $C_{k,n}$ for certain $k, n \in \mathbb{N}$.
- **There are no such k and l.**
  $\vdots$
  ⟨construction of a map from $\mathbb{N}$ to C and proof that it is an isomorphism⟩
  $\vdots$
  Hence C is isomorphic to $(\mathbb{N}, +, 0)$.

Maybe the first thought that comes in mind is to require the decidability of the equivalence relation. But this is not enough to justify the case distinction (see Counterexample 1). Therefore we have split the theorem in two and we have considered the parts separately:

- Every cyclic monoid generated by an element c such that there are $k < l$ with $c^k = c^l$ is isomorphic to $C_{n,m}$ for certain $n, m \in \mathbb{N}$.
- Every cyclic monoid generated by an element c such that there are no $k < l$ with $c^k = c^l$ is isomorphic to $(\mathbb{N}, +, 0)$.

To prove the first part, it seems enough at first sight to require the decidability of the equivalence relation, because when we know there are $k < l$ with $c^k = c^l$, it enables us to conclude there is a smallest pair with that property, i.e.

Statement 1 *There are $k < l$ such that $c^k = c^l$ and for any $k_1 < l_1$ if $(k_1, l_1) < (k, l)$, then $c^{k_1} \neq c^{l_1}$.*

However, because we have to prove a strong version of injectivity of the function constructed in the proof, we would like to have a stronger assumption at our disposal:

Statement 2 *There are $k < l$ such that $c^k = c^l$ and for any $k_1 < l_1$: if $(k_1, l_1) < (k, l)$, then $c^{k_1} \# c^{l_1}$.*

Or equivalently

Statement 3 *There are $k < l$ such that $c^k = c^l$ and for any $k_1 \neq l_1$: if $(k_1, l_1) < (k, l)$, then $c^{k_1} \# c^{l_1}$.*

This seems provable from the decidability of the apartness relation. To make it easy on ourselves, we have not assumed the decidability of the apartness relation and proven this property, but we have assumed this property immediately.

The proof of the second part is now straightforward. So the theorem we have proven eventually in Coq is:

Theorem 8.3.4

Part I *Every cyclic monoid generated by an element c such that there are $k < l$ with $c^k = c^l$ and for all $k_1 \neq l_1$: if $(k_1, l_1) < (k, l)$, then $c^{k_1} \# c^{l_1}$, is isomorphic to $C_{n,m}$ for certain $n, m \in \mathbb{N}$.*

Part II *Every cyclic monoid generated by an element c such that for all $k < l$: $c^k \# c^l$, is isomorphic to $(\mathbb{N}, +, 0)$.*

8.3.3 Stripping Type valued universal quantifications

As discussed in [21], for computational reasons the C-CoRN library has two types for propositions: `Prop` and `CProp`. At first `CProp` was a synonym for `Set`, and `CSetoid` was of type `Set`. But to allow, for example, the subsetoid of partial function over a setoid, `CSetoid` was changed to have type `Type`. Because of consistency reasons this forced `CProp` to change to `Type` too. We work in this way with synonyms to distinguish data-types (in `Set`, `Type`) from propositions (in `Prop`, `CProp`). This difference is crucial for a good rendering.

Right from the start the rendering of propositions in Helm gave problems. Because the XML exportation unfolded the type `CProp`, many propositions were presented like definitions. But even when the exportation module had been improved such that explicit casts to `CProp` were reflected in the XML, some of the propositions were still printed wrongly. The cause of these errors appeared not to lie in the exportation module or in any part of Helm. The problematic propositions appeared simply not to have type `CProp`, but to have necessarily type `Type` instead.

This is the Π-rule for types in Coq:

$$\frac{\Gamma \vdash A : \texttt{Type}_i \quad i \leq k \quad \Gamma, x : A \vdash B : \texttt{Type}_j \quad j \leq k}{\Gamma \vdash \Pi x : A.B : \texttt{Type}_k}$$

Now, `CProp` is `Type`. This means that `CProp` is $\texttt{Type}_j$ for a fixed – but to us unknown – number j. The carrier of a `CSetoid` has type `Type`. But opposed to `CProp`, this `Type` is not fixed. We allow the carrier of a `CSetoid` to have type $\texttt{Type}_i$ for *any* number i. A statement like

```
Lemma ap:  forall (S:CSetoid)(x y:S), x[#]y:  Type.
```

can be understood as follows. Take an arbitrary `CSetoid` S. The carrier of S is of type $\texttt{Type}_i$ for a certain number i. The sort of `ap` is now $\texttt{Type}_k$ for a certain number k. We can not force the sort of `ap` (i.e. type of its type) to be `CProp`, or in other words: we cannot force k to be j, because we have taken an *arbitrary* `CSetoid`. Once this lemma has been proven and the section has been closed, we can instantiate S with any `CSetoid`, also with ones of type $\texttt{Type}_i$ where $i > j$. So the idea to have all data-types in type `Type` or `Set` and all propositions in type `Prop` or `CProp`, cannot be realised with the current definitions of C-CoRN.

To see our lemmas as lemmas and not as definitions in Helm, we have applied a "trick" in the Coq file. Whenever we see a universally quantified computationally relevant proposition, we have put this lemma in a section and we have stripped of all quantifications and made them into section variables until the remaining lemma has type `CProp`. So for lemma `ap` this would be:

```
Variable S: CSetoid.

Lemma ap':  forall (x y:S), x[#]y:  CProp.
```

After the section has been closed, `ap` and `ap'` have the same sort: `Type`. The CICML exportation, however, is aware of the sections in Coq and exports these lemmas to different terms. The sort of the lemmas from within the sections is maintained. Thus, because the sort of `ap'` is the section is `CProp`, it is rendered as a lemma, and not as a definition.

To come to a more satisfactory solution, it has been proposed to make the type universes in C-CoRN explicit, in such a way that the types of data-types are all lower than `CProp`. This has yet to be explored.

8.4 Automatic generation of notational stylesheets

8.4.1 Introduction

The first implementation for automatic generation of the notational stylesheets has been done by Pietro Di Lena and has clearly been recorded in [54] (in Italian). His work includes the definition of a simple XML language for the user, the definition of an intermediate XML language, used by the system, stylesheets from the user's input to the intermediate level, and meta-stylesheets from CICML to MathML-Content, and from MathML-Content to both HTML and MathML-Presentation. He has also tested his implementation on a small sample of mathematical objects.

Example 8.4.1 *This is the code taken from the file* `set.xml` *written by the user, that defines the notation for intersection.*

```
<Symbols type="HTML">
  <symbol tag="intersect" priority="70" associativity="left">
    <case type="symbol" value="&#199;"/>
    <case type="unicode" value="&#8745;"/>
  </symbol>
</Symbols>

<Symbols type="MathMLPresentation">
  <symbol tag="intersect" value="&#x22C2;" priority="70"
                                           associativity="left"/>
</Symbols>

<Operator
 name  ="INTERSECTION"
 uri   ="cic:/Coq/Sets/Ensembles/Intersection.ind#element(/1/1)"
 cook  ="true"
 arity ="2"
 p-tag ="intersect"
 h-tag ="intersect">
      <Notation type="MathMLContent">
        <m:apply helm:xref="\root-id">
          <i:op tag="intersect"/>
          <i:param id="1" mode="set"/>
          <i:param id="2" mode="set"/>
        </m:apply>
      </Notation>
</Operator>
```

The first two paragraphs have been written on the top of the file and define the appearance of the intersect-tag both for HTML and MathML-Presentation. The last paragraph is the description of the operator. The user has to give a name (`name`*), and the exact place where the operator has been defined (*`uri`*). The intersection has been defined inductively. That makes it necessary to add*

`#element(/1/1)` *to the path. Would the user have wanted to define notation not for the intersection, but for its first constructor, then the last part of the uri should have been* `#element(/1/1/1)`*. Then the user says whether the operator has been defined inside a section (*`cook`*). That is the case here. The number of arguments is 2 (*`arity`*) and there are no implicit arguments. If there were, they could have been taken into account with the help of the hide-attribute. Then the system is told to use the newly defined intersect-tags when appropriate (*`p-tag` *and* `h-tag`*). Finally, the user describes what the MathML-Content should look like. Because there is no pre-defined tag for intersection, the user has to put some effort into this. Note that this description refers to the arguments of the operator.*

From this, stylesheets are made from CICML to MathML-Content and from MathML-Content to HTML and MathML-Presentation, that handle the case that this operator is used inside the section where it is defined as well as the case that it is used outside.

This implementation has appeared very useful. It even covers some unforeseen uses, like the handling of natural language (See Section 8.4.7). It has served as a core for further developments.

Although this implementation was well-considered, a big, thorough test was lacking. So when we tried to use it, we not only ran into some bugs, but we also encountered some situations of which the programmers had not thought. This forced us to carry some changes through. Besides that, we have made some changes and added some ideas purely for the convenience of the user. These adaptations affect the definition of the input XML language, the definition of the intermediate XML language, the meta-stylesheet from CICML to MathMLContent, the meta-stylesheet from MathMLContent to HTML, and the arrangement of the input files.

We have not changed the meta-stylesheet from MathMLContent to MathML-Presentation. Most present-day browsers do not render MathMLPresentation in a satisfactory way. Moreover, the Helm system does not use the generated stylesheets from MathMLContent to MathMLPresentation. It uses some general hand-made stylesheets instead ([56]).

8.4.2 Notation inside and outside sections

As already shown in Example 8.4.1, the attribute `cook` is used to store information about whether the definition is made inside or outside a section. When a definition of a mathematical operator is made inside a section and it is applied outside the section of definition, the CICML of the applied operator looks different than when it would have looked if it were not defined inside a section. Because a section can contain variable declarations, a definition made within a section can rely on those variables, without explicitly mentioning them as arguments of the operator. Is this operator fully applied outside the section, then these variables get *instantiated*. The instantiation of variables is not seen as application.

Example 8.4.2

```
Section S1
     Variable A : CSetoid.
     Variable B : CSetoid.

     Definition product_inside :  CSetoid.
          <definition using A and B>
     Qed.
End S1.

Definition product_outside :  forall (A B:CSetoid), CSetoid.
     <definition using A and B>
Qed.

Section S2.
      Variable C : CSetoid.
      Variable D : CSetoid.

      Lemma iso :
        (isomorphic (product_inside C D) (product_outside C D)).
End S2.
```

In the first section, `S1`*, two variables,* `A` *and* `B` *are declared. Then the definition of product between constructive setoids,* `product_inside`*, is given using these variables. Outside this section, another definition of product between constructive setoids is given (*`product_outside`*). The definitions are used similarly in* `Lemma iso` *in the section* `S2`*.*

The CICML that codes the application of these operation in this lemma differs. The CICML code that codes `(product_inside C D)` *is:*

```
<APPLY>
  <instantiate>
    <CONST uri="cic:/CoRN/algebra/CSetoids/product_inside.con"/>
    <arg relUri="S1/A.var">
      <VAR uri="cic:/CoRN/algebra/CSetoids/S2/C.var"/>
    </arg>
    <arg relUri="S1/B.var">
      <VAR uri="cic:/CoRN/algebra/CSetoids/S2/D.var"/>
    </arg>
  </instantiate>
</APPLY>
```

whereas the CICML code that codes `(product_outside C D)` *is:*

```
<APPLY>
  <CONST uri="cic:/CoRN/algebra/CSetoids/product_outside.con"/>
  <VAR uri="cic:/CoRN/algebra/CSetoids/S2/C.var"/>
  <VAR uri="cic:/CoRN/algebra/CSetoids/S2/D.var"/>
</APPLY>
```

So this information is crucial for the correct generation of the stylesheets.

In the original work of Di Lena [54] the user could say only *whether* a definition has been given inside a section. If this was the case there was no way to say *how many* variables it relied on. This means that it was not possible for the user to refer to these variables when he defined how this operator should be rendered. They were automatically rendered not unlike implicit arguments. This is a very practical solution, because the variables are never instantiated when one uses the definition in its defining section, and now it sufficed to give just one description of the rendering, that was used to generate the templates both for use inside the defining section and for use outside.

The result, however, was not very satisfactory, because sometimes one wants to involve arguments in the rendering that are instantiations, and not applications. The choice to let a definition depend on variables seems to be influenced more by convenience and programming style, than by thoughts of the rendering. So we have changed the cook-attribute to have a numerical value: the number of section variables a definition depends on. This also affects the meaning of the hide-attribute. At first the hide-attribute indicated the number of implicit arguments, now it indicates the number of implicit instantiations and arguments. (Implicit objects are always assumed to precede the explicit ones). Note that when the value of the cook-attribute is bigger than the value of the hide attribute –that is: not all section variables a definition depends on are seen as implicit– the user has to describe two different renderings: one for inside the section and one for outside, marked respectively by `cooked = "false"` and `cooked = "true"` (default).

Example 8.4.3 *The following describes a rendering for* `product_inside`*, assuming that* `\A`*,* `\B`*, and* `\times` *have already been defined. (MathMLPresentation has been omitted.):*

```
<Operator
   name    =  "PRODUCT"
   uri     =  "cic:/CoRN/algebra/CSetoids/product_inside.con"
   cook    =  "2"
   arity   =  "0" >
    <Notation type="MathMLContent" cooked="false">
      <m:apply helm:xref="\root-id">
        <i:op tag="times" definitionURL="\uri"/>
        <i:op tag="ci" definitionURL="\uri" bsymbol="A">A</i:op>
        <i:op tag="ci" definitionURL="\uri" bsymbol="B">B</i:op>
      </m:apply>
    </Notation>

    <Notation type="MathMLContent">
      <m:apply helm:xref="\root-id">
        <i:op tag="times" definitionURL="\uri"/>
        <i:param id="1"/>
        <i:param id="2"/>
```

```
      </m:apply>
    </Notation>

    <Notation type="HTML" cooked="false">
      \A \times \B
    </Notation>

    <Notation type="HTML">
      <o:param id="1"/>\times<o:param id="2"/>
    </Notation>
</Operator>
```

Now, `product_inside` *will be printed inside the section of definition as* $A \times B$*, and if the variables are instantiated outside the section, say by* C *and* D*, it is rendered as* $C \times D$*.*

8.4.3 Non-applied mathematical operators

Let us now have a look at another simple example.

Example 8.4.4 *Assuming the plus-tag has already been defined for HTML and MathMLPresentation, the following defines the rendering for the addition operator:*

```
<Operator
   name    =  "PLUS"
   uri     =  "cic:/Coq/Init/Peano/plus.con
              | cic:/Coq/ZArith/BinInt/Zplus.con"
   arity   =  "2"
   c-tag   =  "plus"
   p-tag   =  "plus"
   h-tag   =  "plus"/>
```

From the information given in Example 8.4.4 all stylesheets are generated that take care of the expected rendering of the addition operator, provided that *it acts on two arguments*. It will not apply, for example, to a term like `(commutative plus)`, because here `commutative` is applied, and `plus` is not. But even though `plus` is non-applied, we would like to render this term as "+ is commutative". So we would like the stylesheets to apply.

To this end, we have extended the set of attributes of `Operator` with `object`, which has a Boolean value. Default value is `object = "false"`, that will generate the same stylesheets as before; when `object = "true"` also stylesheets that apply in the case that the operator does not get applied. We have extended the set of attributes of `Notation` accordingly: to describe the notation in the non-applied case, add `arity_0 = "true"`.

Observe that in the worst case, i.e. we want the stylesheets for the non-applied case and our operator is defined within a section (See Section 8.4.2), it can occur that the user has to give four descriptions for each type of notation (MathMLContent, MathMLPresentation, HTML).

8.4.4 Non-linear transitions

Many times there is a 1-1 correspondence between the arguments in the CIC representation of a mathematical operator and the arguments in its final rendering. If this is the case, we call the transition *linear*. For instance, we would like to see "`(plus x y)`" as "$(x+y)$". In both representations the addition has two arguments and they are linked with each other in the obvious way.

It also quite often occurs that the final rendering contains fewer arguments than the CIC representation. This happens for example in the case of strictly implicit arguments: one or more arguments are not printed, but they could be recovered from the arguments that are shown. For readability, we have chosen to also allow non-strictly implicit arguments: We sometimes hide arguments that can not be recovered from the remaining information. The function `cf_div`, for example, expects not only a field, a numerator and a denominator, but also a proof that the denominator is apart from zero. We do not need to record the field, because we can restore it from the types of the numerator and the denominator. So we take this as a strictly implicit argument. But also the proof that the denominator is apart from zero we throw away, because this is usually not seen as an argument in a standard mathematical text. This proof cannot be restored from the remaining information. It is not obvious that this is the right thing to do.

In our sample, it happens, though rarely, that we would like the final rendering to have more arguments than the CIC representation, i.e. we would like to see an argument more than once. For example the semi-group of constructive setoid functions, `FS_as_CSemiGroup`, takes only one element, that we like to render twice: "`(FS_as_CSemiGroup S)`" should be printed as "$S \to S$". It was not possible to generate stylesheets to take care of this.

There was a technical problem that restrained us from generating stylesheets for rendering `(FS_as_CSemiGroup S)` appropriately. One of the generated stylesheets computes the length of the output string. In this stylesheet the length of each argument is stored in a variable. Such a variable took the name of the serial number in the CIC representation of the argument whose length it holds. So, in our example, the variable that stores the length of `S`, would have been called `charcount_param_1`, because it refers to the first argument of `FS_as_CSemiGroup`. But because we want to see this argument twice, a variable with this name was also defined twice. It is not allowed in XSLT to define a variable in the scope of another variable with the same name.

We have solved this by giving the variable the name of the serial number in the HTML presentation of the argument which length it holds.

8.4.5 Notation for variables

Although it was possible to generate stylesheets that took care of constants, inductively defined objects and even constructors, there was no way to do that for variables, too. But according to our experience, a nice rendering of variables increases the readability. Therefore we have adapted the meta-stylesheets to

handle also variables.

8.4.6 Central definition of symbols

As shown in Example 8.4.1, the files from which the meta-stylesheets take their data to generate the stylesheets, starts with the definitions of the required tags: HTML tags for both symbol and unicode font and MathMLPresentation tags. These tags are used in the files that hold the stylesheets from MathMLContent to HTML and from MathMLContent to MathMLPresentation. So when we look in the file `html_set.xsl`, we see the following variables:

```
<xsl:variable name="set_intersect">
 <xsl:choose>
  <xsl:when test="$UNICODEvsSYMBOL='symbol'">&#xC7;</xsl:when>
  <xsl:when test="$UNICODEvsSYMBOL='unicode'">&#x2229;</xsl:when>
  <xsl:otherwise>???</xsl:otherwise>
 </xsl:choose>
</xsl:variable>
<xsl:variable name="set_intersect_priority" select="70"/>
<xsl:variable name="set_intersect">
 <xsl:choose>
  <xsl:when test="$UNICODEvsSYMBOL='symbol'">&#xC7;</xsl:when>
  <xsl:when test="$UNICODEvsSYMBOL='unicode'">&#x2229;</xsl:when>
  <xsl:otherwise>???</xsl:otherwise>
 </xsl:choose>
</xsl:variable>
<xsl:variable name="set_intersect_priority" select="70"/>
```

(Note that the user has given the codes for the unicode and the symbol sign in decimal notation in this case, and that Helm has made hexadecimal notation out of it.) These variables are referred to only inside this file, as the names of the variables reflect.

The fact that the tags are only defined for the file they have been put in, is often a disadvantage. Sometimes one needs the same symbol in two files and hence one has to define the same tags several times. To overcome this, we have put all tag definitions in one file, `HTML_SYMBOLS.xml`, and we have included this file – after a little transformation – in every other .xml-file.

So, an advantage of this approach is that the user needs only to define his symbols once. Or maybe even better: if we ourselves put the most common symbols in this file, say all LaTeX symbols with the names used in LaTeX, there would be hardly any need for the user to define any symbol at all.

There are also several disadvantages. Firstly, to include the symbols-file in every other .xml-file, we have used the XInclude package, which is an extension of XML that is not universally implemented. Secondly, at first an html_*.xsl-file contained only the tag definitions it really needed, and now an html_*.xsl-file contains *all* tag definitions. So the files have become bigger.

Both these disadvantages could probably be remedied by including the symbols in a much later stage, in an XSLT file instead of in the XML files. To do this, another adaptation of the meta-stylesheets is required.

8.4.7 Natural language generation

Although the generated stylesheets had only been used to translate CIC-expressions to their usual mathematical symbolic rendering, for instance to replace "`plus`" by "+", the machinery was able to generate stylesheets to produce a bit of natural language too. So now the generated stylesheets transform "`(commutes plus nat)`" into "+ is commutative on ℕ", with links from "+" to the definition of "`plus`", from "ℕ" to the definition of "`nat`" and from "is", "commutative" and "on" to the definition of "`commutes`". To get the links right, we have made extensive use of the attribute "`bsymbol`", which helps connecting the presentation to the content level. This is the XML description for the rendering of `commutes`:

```
<Operator
   name   =  "IS COMMUTATIVE"
   uri    =  "cic:/CoRN/algebra/CSetoids/commutes.con"
   cook   =  "1"
   hide   =  "0"
   arity  =  "1">
    <Notation type="MathMLContent">
      <m:apply>
        <i:op tag="csymbol" bsymbol="is,commutative,on">
          is_commutative
        </i:op>
        <i:param id="1"/>
        <i:param id="2"/>
      </m:apply>
    </Notation>

    <Notation type="HTML">
      <o:param id="2"/>
       \is  \commutative  \on  
      <o:param id="1"/>
    </Notation>

</Operator>
```

where `\is`, `\commutative` and `\on` have been defined in the file `HTML_SYMBOLS.xml` as follows:

```
<symbol tag="commutative" value="commutative"/>
```

```
<symbol tag="is" value="is"/>
<symbol tag="on" value="on"/>
```

Besides that this is a rather onerous way to do something as simple as to print a part of a sentence, a drawback is that we have to use the non-breakable space ` `, since all breakable spaces are removed as part of the XSLT process. This severely limits the possibilities the user's browser has to break lines and in some cases leads to unbalanced line lengths or even a line being longer than the browser's window is wide.

At the moment, we have only generated natural language for mathematical operators. Logical operators are still presented symbolically. So we see "$\wedge$" and "$\rightarrow$", instead of "and" and "implies". Maybe these would have to change too. Then our presentation would be more uniform in style.

8.5 Results

A total of 21 pages of IDA has been incorporated in Helm in a fully or partially formalised way. These pages treat binary operations and monoids and consist of:

- 10 pages explaining the theory;
- 9 pages of examples;
- 2 pages of remarks.

Most lemmas, definitions and examples in these pages have been replaced by the Helm rendering of their formal counterparts in C-CoRN, for which notation has been provided. Following links in these formal objects results in entering the C-CoRN library as rendered by Helm, giving the user the possibility to track all definitions and to explore all proofs.

At the same time, the structure and style of IDA have been preserved, including the functionality of the lower frame of the theory-pages and the drop down menus in the middle of those, allowing the user to navigate through the document.

Figure 8.6 compares the page that defines a monoid in our course notes (on the left hand side) with the corresponding page in IDA (on the right hand side). Note the differences between C-CoRN and IDA for the statement of uniqueness of a unit and the definition of a monoid. One of the reasons for the increased verbosity of the formal definitions is that the formal library lacks or does not exploit a few notions that help to make the statements more concise. For instance, it would be possible to define the notion of uniqueness over a type T as a property over T. Then we would use this property to state the lemma of uniqueness of the unit element. This is not normally done when working in Coq since Coq unification does not automatically expand this kind of general notions, making proofs much more cumbersome for the user. The

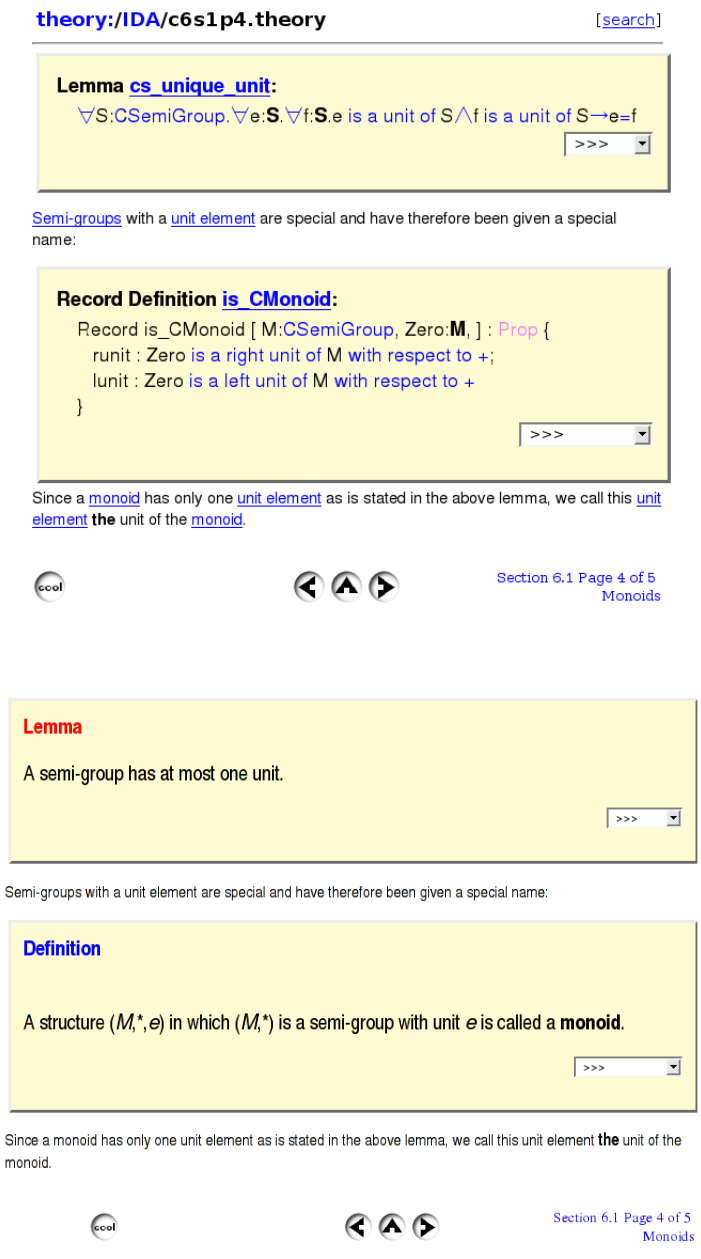


Figure 8.6: Comparison between our course notes (on top) and IDA.

same phenomenon appears in the definition of a monoid. Instead of relying on the notion of unit, in C-CoRN it is preferred to explicitly state that `Zero` is

both a left and a right unit, since in Coq the constituents of a conjunction are not automatically derived from the conjunction when appropriate.

If the general notions had been used, we believe that the differences between the formal and informal versions would be less relevant and a bit of extra notation would provide a rendering that is not more cumbersome than the humanly provided counterpart.

8.6 Discussion

8.6.1 Context dependent rendering

The rendering of mathematics in the document does not have a notion of before and after: mathematical objects are printed the same, independently of where they occur in the document. This does not seem to be always what one wants. In the theory of semi-groups, for instance, first the notion of being a unit is defined and after that the uniqueness of the unit in semi-groups is proven. So when we state the uniqueness of the unit, we would like to speak about "*a* unit", because the uniqueness has not yet been established:

Lemma 8.6.1 *∀S:CSemiGroup.∀e:S.∀f:S. e is a unit of S∧f is a unit of S→e=f*

(Note that this does not come close to the way it is formulated in the original IDA; there it simply reads "A semi-group has at most one unit".) But after we have proven the uniqueness, we would like to see "*the* unit" in all following uses.

What is more, web documents do not have to be linear like a book. It is well thinkable that there is more than one route through the document. This clouds the notion of before and after.

It has also been argued that this is not really a problem, because differences in the English language – like "a" and "the" here – reflect also a difference in the mathematical meaning: "a unit" would denote a relation and "the unit" would denote a function. So, it has been proposed to use "a" whenever we see the relation, and to use "the" whenever we see the function. However, it is not clear that the correspondence between the English language and the mathematical meaning is as straightforward as suggested here. It is not a mistake to use the relation even after uniqueness has been established, but to use the indefinite article in English while we know the uniqueness, seems highly unusual.

8.6.2 Moving mathematical objects

The order in which the Helm system forces the user to create the document, i.e. first formalise the mathematics and only after that describe its rendering is quite inconvenient. Because in this fashion the mathematics is completely separated from the description, there has to be another a way to connect the two. In Helm this link is made by the user, who writes in the XML description explicitly where the object can be found. This means that whenever an object in the library moves, the user has to alter its XML description.

And objects do move, because of the way we work with C-CoRN. First, when we start to develop a theory, all objects are in our personal `devel` directory. And only later, when the development has been completed, everything is transferred to the fast core of C-CoRN. So, the Helm system imposes us to either write the XML description only after we have transferred everything, or to change all descriptions when we move the objects.

A possible answer to this is the tool developed by Trusted Logic ([35]), which generates all kinds of notational information out of the proof-script and the texts surrounding it.

8.6.3 Automatic proving

In line with the philosophy described in Section 8.1.4, we make no major transformation on the input lambda term (the proof object) before presenting it. It is obvious that with a little effort we could improve the presentation by suppressing (or removing) a lot of detail. A typical example are proof objects that are found by an automatic decision procedure: these are usually extremely verbose and complex proofs that humans do not want to read. Not showing these subproofs' details by default would probably work well. However, in the proof object as such we can't trace the application of an automation tactic, so this would require a major transformation of the term before rendering it. A particularly fortunate case of an automation tactic in Coq is that of a *reflexive tactic*. The crucial point is that it does not create a huge proof term. Instead it encodes the goal to be proven as a syntactic object and then applies a generic lemma to an argument that solves the goal by mere computation. So, to detect a reflexive tactic it would be sufficient to recognise (in the stylesheets) an application of the generic lemma and hide its rendering.

8.6.4 Use of a formal library: economy vs. coherence

This research has not only been a test case for Helm and the way it presents formal mathematics, but it has also been a test for the use of a formal library, especially C-CoRN, in a mathematical document. The choice to use the mathematics in the library has some huge advantages: you can use everything that is already in the library. In our case, not much from what we needed was present in C-CoRN when we started: only about 20% from the definitions, lemmas and examples we have used in formal form in our final document. That seems a bit disappointing. But it has been very useful to us any way, because many items that were missing in the library were fairly easy to formalise from lemmas that were present. And by adding items to the library, we have made a contribution to future use.

So, in short, it is beneficial to use a library because it is *economical*: if something has been defined or proven, it can be used over and over again. Nothing has to be done more than once.

However, there are also disadvantages of using a library for this purpose. It seems to neglect the *coherence* of the document and it seems to destroy – in a

weak sense – the existing coherence of the library.

A mathematical document usually answers a strict requirement: objects and lemmas are not to be used before they have been defined or proven. Of course, sometimes a lemma is used and is only proven later at a more suitable place, but this is mostly announced in the surrounding text and this can be seen as the use of a local assumption, instead of premature use of a lemma. But using an object or a lemma that is unfamiliar by the reader without any explanation, can be seen as a mistake. By using a library, we have paid no attention to this kind of logical coherence in the document. In the document we have put references to the library and it could well be that the order of the references does not meet the logical order of the corresponding items in the library. If we had not used a library, we could have taken the coherence into account. We could have started from scratch by making Coq files that follow the structure of the document. Because Coq guards the logical coherence of the mathematics, the logical coherence in the document would be guaranteed too.

The fear for logical mistakes *between* items, as opposed to logical mistakes *within* items (proofs and definitions), is not imaginary. In the ten pages of IDA that we have looked at, the logical coherence has been violated several times.

Also C-CoRN has some kind of coherence: not only a formal coherence, but also an informal coherence, introduced by the documentation. Here by "documentation" is meant the text surrounding the formal objects; the part of a Coq file that is not checked. Most of the time this text explains some difficulties or particularities of the formalisation, but it also occurs, though not often, that the documentation and the division of the file reflect the original goal of the formalisation. In these cases it is very hard to add similar lemmas or even to use existing lemmas without changing the documentation. For example if one wants to add a lemma to a section and the documentation reads "The last lemma of this section is about ..." or one has used a lemma which is accompanied by a text saying that it is useful for a particular other lemma, one has to update the documentation. Although a good, informal explanation of the mathematical line of thought makes C-CoRN nice to read, there seems to be some tension between having a documentation geared to the mathematical content, and the reusability of the library.

Bibliography

[1] Agda: An interactive proof editor. http://agda.sf.net.

[2] Andrea Asperti, Herman Geuvers, Iris Loeb, Lionel Elie Mamane, and Claudio Sacerdoti Coen. An interactive algebra course with formalised proofs and definitions. deliverable, European project IST-2001-33562 Mathematics On the Web: Get it by Logic and Interfaces (MoWGLI), 2005.

[3] Andrea Asperti, Herman Geuvers, Iris Loeb, Lionel Elie Mamane, and Claudio Sacerdoti Coen. An interactive algebra course with formalised proofs and definitions. In Michael Kohlhase, editor, *MKM*, volume 3863 of *Lecture Notes in Computer Science*, pages 315–329. Springer, 2006.

[4] Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages.* Cambridge University Press, 1998.

[5] Andrea Asperti, Ferruccio Guidi, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. A content based mathematical search engine: Whelp. In Jean-Christophe Filliâtre, Christine Paulin-Mohring, and Benjamin Werner, editors, *TYPES*, volume 3839 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2006.

[6] Andrea Asperti and Harry G. Mairson. Parallel beta reduction is not elementary recursive. In *POPL*, pages 303–315, 1998.

[7] Andrea Asperti, Luca Padovani, Claudio Sacerdoti Coen, Ferruccio Guidi, and Irene Schena. Mathematical knowledge management in HELM. *Ann. Math. Artif. Intell.*, 38(1-3):27–46, 2003.

[8] Henk Barendregt. Towards an interactive mathematical proof language. ftp://ftp.cs.ru.nl/pub/CompMath.Found/mathmode.pdf.

[9] Henk Barendregt. *The lambda calculus: its syntax and semantics.* North-Holland Publish. Comp., 1981.

[10] Henk Barendregt and Herman Geuvers. Proof-assistants using dependent type systems. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1149–1238. Elsevier and MIT Press, 2001.

[11] H.P. Barendregt, J.A. Bergstra, J.W. Klop, and H. Volken. Degrees, reductions and representability in the lambda calculus. Peprint 22, Department of Mathematics, Utrecht University, 1976. The 'blue preprint'.

[12] Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. Sharing in the weak lambda-calculus. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles*, volume 3838 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2005.

[13] Roel Bloo and Herman Geuvers. Explicit substitution on the edge of strong normalization. *Theor. Comput. Sci.*, 211(1-2):375–395, 1999.

[14] Samuel R. Buss. The undecidability of k-provability. *Annals of Pure and Applied Logic*, 53(1):75–102, 1991.

[15] Alessandra Carbone. Interpolants, cut elimination and flow graphs for the propositional calculus. *Ann. Pure Appl. Logic*, 83(3):249–299, 1997.

[16] Alessandra Carbone. Duplication of directed graphs and exponential blow up of proofs. *Ann. Pure Appl. Logic*, 100(1-3):1–67, 1999.

[17] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33/34:346–366, 839–864, 1932/33.

[18] Arjeh M. Cohen, Hans Cuypers, and Hans Sterk. *Algebra Interactive!: learning algebra in an exciting way.* Springer, 1999.

[19] The Coq proof assistant. http://coq.inria.fr.

[20] Constructive Coq Repository at Nijmegen. http://c-corn.cs.kun.nl.

[21] Luís Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications.* PhD thesis, Katholieke Universiteit Nijmegen, 2004.

[22] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-CoRN, the Constructive Coq Repository at Nijmegen. In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *MKM*, volume 3119 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2004.

[23] Luís Cruz-Filipe and Bas Spitters. Program extraction from large proof developments. In David A. Basin and Burkhart Wolff, editors, *TPHOLs*, volume 2758 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 2003.

[24] Vincent Danos and Laurent Regnier. The structure of the multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[25] René David and Delia Kesner. An arithmetical strong-normalisation proof for reduction modulo proof-nets. Draft.

[26] Frederic Brenton Fitch. *Symbolic Logic: an Introduction.* New York: Ronald Press Company, 1952.

[27] Gerhard Gentzen. Untersuchungen über das logische Schliessen. In M.E. Szabo, editor, *Collected Papers of Gerhard Gentzen.* North-Holland Publishing Company, 1969.

[28] Herman Geuvers and Iris Loeb. Natural deduction via graphs: Formal definition and computation rules. To appear in MSCS.

[29] Herman Geuvers and Iris Loeb. From deduction graphs to proof nets: Boxes and sharing in the graphical presentation of deductions. In Rastislav Královič and Pawel Urzyczyn, editors, *MFCS*, volume 4162 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2006.

[30] Herman Geuvers and Iris Loeb. Deduction graphs with universal quantification. In Ian Mackie and Detlef Plump, editors, *Pre-Proceedings of TERMGRAPH 2007*, pages 66–80, 2007.

[31] Herman Geuvers and Rob Nederpelt. Rewriting for fitch style natural deductions. In Vincent van Oostrom, editor, *RTA*, volume 3091 of *Lecture Notes in Computer Science*, pages 134–154. Springer, 2004.

[32] Herman Geuvers, Randy Pollack, Freek Wiedijk, and Jan Zwanenburg. A constructive algebraic hierarchy in Coq. *J. Symb. Comput.*, 34(4):271–286, 2002.

[33] Herman Geuvers, Freek Wiedijk, and Jan Zwanenburg. A constructive proof of the fundamental theorem of algebra without using the rationals. In Paul Callaghan, Zhaohui Luo, James McKinna, and Robert Pollack, editors, *TYPES*, volume 2277 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2000.

[34] Mariusz Giero and Freek Wiedijk. MMode, a mizar mode for the proof assistant Coq. Report NIII-R0333, University of Nijmegen, 2003.

[35] Eduardo Gimenez. Validation 2: Smart card security. Mowgli deliverable, European project IST-2001-33562 Mathematics On the Web: Get it by Logic and Interfaces (MoWGLI), 2005.

[36] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[37] Jean-Yves Girard. Quantifiers in linear logic II. In Giovanna Corsi and Giovanni Sambin, editors, *Atti della Congresso: Nuovi Problemi della Logica e della Filosofia della Scienza, Vol. 2*, pages 79–89. Bologna: CLUEB, 1991.

[38] Anjolina Grisi de Oliveira and Ruy J. G. B. de Queiroz. Geometry of deduction via graphs of proofs. In *Logic for concurrency and synchronisation*, pages 3–88. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[39] helm: Hypertextual Electronic Library of Mathematics. http://helm.cs.unibo.it.

[40] The HOL Light theorem prover. http://www.cl.cam.ac.uk/users/jrh/hol-light.

[41] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, Inc., New York, N.Y., 1980.

[42] HyperText Markup Language (HTML). http://www.w3.org/MarkUp.

[43] Isabelle. http://isabelle.in.tum.de.

[44] Stanisław Jaśkowski. *Księga Pamiątkowa Pierwszego Zjazdu Matematycznego*, page 36, 1929. Kraków.

[45] Stanisław Jaśkowski. On the rules of suppositional formal logic. In Storrs McCall, editor, *Polish Logic 1920-1939*, pages 232–258. Clarendon Press, Oxford, 1967.

[46] Delia Kesner and Stephane Lengrand. Extending the explicit substitution paradigm. In Jürgen Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 407–422. Springer, 2005.

[47] Zurab Khasidashvili. Optimal normalization in orthogonal term rewriting systems. In Claude Kirchner, editor, *RTA*, volume 690 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 1993.

[48] S. C. Kleene. λ-definability and recursiveness. *Duke Math. J.*, 2:340–353, 1936.

[49] Jan Willem Klop. *Combinatory Reduction Systems*. PhD thesis, University of Utrecht, 1980.

[50] W. Kneale. The province of logic. *Contemporary British Philosphy, third series*, pages 235–261, 1956.

[51] Yves Lafont. Interaction nets. In *POPL*, pages 95–108, 1990.

[52] Yves Lafont. Introduction to interaction nets, 1998. Manuscript.

[53] John Lamping. An algorithm for optimal lambda calculus reduction. In *POPL*, pages 16–30, 1990.

[54] Pietro Di Lena. Generazione automatica di stylesheet per notazione matematica. Master's thesis, University of Bologna, 2002.

[55] Jean-Jacques Lévy. *Réductions Correctes et Optimales dans le lambda-calcul.* PhD thesis, Université Paris VII, 1978.

[56] Iris Loeb. Presentational stylesheets. Mowgli deliverable, European project IST-2001-33562 Mathematics On the Web: Get it by Logic and Interfaces (MoWGLI), 2003.

[57] Ian Mackie. Linear logic *with* boxes. In *LICS*, pages 309–320, 1998.

[58] Ian Mackie. Interaction nets for linear logic. *Theor. Comput. Sci.*, 247(1-2):83–140, 2000.

[59] John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theor. Comput. Sci.*, 228(1-2):175–210, 1999.

[60] W3C Math Home. http://www.w3.org/Math.

[61] Storrs McCall, editor. *Polish Logic 1920-1939.* Clarendon Press, Oxford, 1967.

[62] Robin Milner. Axioms for bigraphical structure. Technical report UCAM-CL-TR-581, Computer Laboratory, University of Cambridge UK, 2004.

[63] Mizar. http://www.mizar.org.

[64] Eugenio Moggi. Computational lambda calculus and monads. Technical report ECS-LFCS-88-66, LFCS University of Edinburgh UK, 1988.

[65] MoWGLI; Mathematics on the Web: Get it by Logic and Interfaces. http://mowgli.cs.unibo.it.

[66] Rob P. Nederpelt. *Strong Normalization in a typed lambda calculus with lambda structured types.* PhD thesis, Technical University Eindhoven, 1973.

[67] PRL Project: "Proof/Program Refinement Logic". http://www.cs.cornell.edu/Info/Projects/NuPRL.

[68] OMDoc: Open Mathematical Documents. http://www.mathweb.org/omdoc.

[69] Francis Jeffry Pelletier. A history of natural deduction and elementary logic textbooks. In J. Woods and B. Brown, editors, *Logical Consequence: Rival Approaches, Vol. 1, Proceedings of the 1999 Conference of the Society of Exact Philosophy*, pages 105–138. Oxford: Hermes Science Pubs, 2001.

[70] Dag Prawitz. *Natural Deduction: a proof-theoretical study.* Stockholm: Almqvist och Wiksell, 1965.

[71] Dag Prawitz. Ideas and results in proof theory. In Jens Erik Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in logic and the foundations of mathematics*, pages 235–309. North-Holland Publishing Company, 1971.

[72] Quintijn Puite. Proof nets with explicit negation for multiplicative linear logic. Preprint 1079, Department of Mathematics, Utrecht University, 1998.

[73] PVS Specification and Verification System. http://pvs.csl.sri.com.

[74] Femke van Raamsdonk. *Confluence and Normalisation for Higher Order Rewriting.* PhD thesis, Vrije Universiteit Amsterdam, 1996.

[75] Henrik Kragh Sørensen. Exceptions and counterexamples: Understanding Abel's comment on Cauchy's theorem. *Historia Mathematica*, 32(4), 2005.

[76] Morten Heine Sørensen. *Normalization in Lambda-Calculus and Type Theory.* PhD thesis, University of Copenhagen, 1997.

[77] Claudio Sacerdoti Coen. Exportation module. Mowgli deliverable, European project IST-2001-33562 Mathematics On the Web: Get it by Logic and Interfaces (MoWGLI), 2002.

[78] S.J. Shoesmith and T.J. Smiley. *Multiple-Conclusion Logic.* Cambridge University Press, 1978.

[79] François-Régis Sinot. *Stratégies Efficaces et Modèles d'Implantation pour les Langages Fonctionnels.* PhD thesis, École Polytechnique, 2006.

[80] Richard Statman. *Structural Complexity of Proofs.* PhD thesis, Stanford University, 1974.

[81] Richard Statman. The typed lambda-calculus is not elementary recursive. *Theor. Comput. Sci.*, 9:73–81, 1979.

[82] W. W. Tait. Infinitely long terms of transfinite type I. In J. N. Crossley and M. A. E. Dummet, editors, *Formal Systems and Recursive Functions, Proceedings of the eighth Logic Colloquium, Oxford, 1963*, Studies in Logic and the Foundations of Mathematics, pages 176–185. Amsterdam: North-Holland Publishing Company, 1965.

[83] Terese. *Term Rewriting Systems.* Cambridge University Press, 2003.

[84] Kazushige Terui. Light affine calculus and polytime strong normalization. In *LICS*, 2001.

[85] Richmond H. Thomason. *Symbolic Logic: an Introduction.* New York: Macmillan, 1970.

[86] Lorenzo Tortora de Falco. *Réseaux, cohérence et expériences obsessionelles.* PhD thesis, Université Paris VII- Denis-Diderot, 2000.

[87] Anne S. Troelstra. *Lectures on Linear Logic.* CSLI lecture notes; no.29. Menlo Park, CA etc., 1992.

[88] Freek Wiedijk. The mathematical vernacular. http://www.cs.ru.nl/ freek/notes/mv.ps.gz, 2002.

[89] Extensible Markup Language (XML). http://www.w3.org/XML.

[90] XSL Transformations (XSLT). http://www.w3.org/TR/xslt.

Index

Appendix A

Fitch-Style Flag Deductions

A.1 The systems FD, FD$^-$, and FD$_a$

This section contains an informal overview of Fitch-style flag deduction. We will only treat Fitch-style flag deduction for first-order predicate logic: The rules for proposition logic can easily be inferred from that. See also [26], [85] and [31].

The language for Fitch-style first-order flag deductions is Pred, which is already discussed in Chapter 7.

Figure A.1 gives an example of a Fitch-style flag deduction. It consists of numbered lines followed by a number of vertical bars, which indicate *subordinate proofs*. Then follows a formula (the *label* of the line), and finally there might be a motivation. For example, the lines 2 to 7 form a subordinate proof of the

1	$\forall y.\mathtt{P}(y)\rightarrow\forall y.\mathtt{Q}(y)$		
2	$\forall y.\mathtt{P}(y)$		
3	$\forall y.\mathtt{Q}(y)$		$\Rightarrow$E, 1, 2
4	u	$\mathtt{Q}(u)$	$\forall$E, 3
5		$\mathtt{P}(u)$	
6		$\mathtt{P}(u)\rightarrow\mathtt{Q}(u)$	$\Rightarrow$I, 5, 4
7	$\forall x.\mathtt{P}(x)\rightarrow\mathtt{Q}(x)$		$\forall$I, 6

Figure A.1: A Fitch-style flag deduction.

lines 1 to 7, and so does line 5. We also say that the *subordinate proof at line* 3 are the lines 2 to 7. The subordinate proofs represent some kind of *scoping*: Suppose on line m the motivation refers to line l. Then the subordinate proof at line l includes line m. In that case we say that l *is in scope of* m.

On the lines 1, 2, and 5, we see *flags*: a special construction starting a subordinate proof. These represent hypotheses. Line 4 also starts a subordinate proof and may be seen as a flag of a different kind: One that introduces a free variable.

Line 7 *closes* the subordinate proof that starts at line 4, and line 6 closes the subordinate proof that starts at line 5. The subordinate proofs starting at the lines 1 and 2 are still *open*. We say that the subordinate proof starting at line 2 is the *last open* subordinate proof.

We discuss briefly the various steps of Fitch-style flag deductions (*flag deductions* for short).

Definition A.1.1 (Flags) *A formula A on the first line, called a* flag*, that starts a* subordinate deduction*, is a flag deduction.*

$$\begin{array}{l|l} 1 & \multicolumn{1}{|l}{\underline{A}} \end{array}$$

Given a flag deduction, we may extend it with a new line m, called a flag*, labelled with a formula A. This also starts a* subordinate deduction*.*

$$\begin{array}{l|l} m & \multicolumn{1}{|l}{\underline{A}} \end{array}$$

A flag deduction may also start by putting a variable u on line 1 *and starting a subordinate proof. Similarly, given a flag deduction, we may extend it with a new line m, labelled with the variable u and starting a new subordinate proof, provided that m is not part of a subordinate proof that is already marked with u. We will call u the* eigenvariable *of the subordinate proof and the subordinate proof a* variable subordinate proof

$$\begin{array}{l|l|} 1 & u \end{array} \qquad\qquad \begin{array}{l|l|} m & u \end{array}$$

If the motivation of a line occurring in a variable subordinate proof refers to a line n, n either belongs to the same subordinate proof, or it is a reiteration (Definition A.1.4) of a formula in which u does not occur. The following definitions should be understood to answer this condition.

Definition A.1.2 ($\rightarrow I$) *Given a flag deduction with the last subordinate proof starting at line m, labelled A, and a line k labelled B that is in scope of line m we may extend the deduction with a line l labelled $A \rightarrow B$ with motivation $\Rightarrow I, m, k$, which closes the subordinate proof.*

$$\begin{array}{l|l|l} m & \multicolumn{1}{|l}{\underline{A}} & \\ & \vdots & \\ k & B & \\ & \vdots & \\ l & A \rightarrow B & \Rightarrow I,\ m,\ k \end{array} \qquad\qquad \begin{array}{l|l|l} k & B & \\ & \vdots & \\ m & \multicolumn{1}{|l}{\underline{A}} & \\ & \vdots & \\ l & A \rightarrow B & \Rightarrow I,\ m,\ k \end{array}$$

Definition A.1.3 ($\rightarrow E$) *Given a flag deduction with a line n labelled $A{\rightarrow}B$, and a line m labelled m, we may extend it with a new line k labelled B with motivation $\Rightarrow$ E, under the proviso that both n and m are in scope of k.*

n	$A{\rightarrow}B$		m	A	
	$\vdots$			$\vdots$	
m	A		n	$A{\rightarrow}B$	
	$\vdots$			$\vdots$	
k	B	$\Rightarrow E,\ n,\ m$	k	B	$\Rightarrow E,\ n,\ m$

Definition A.1.4 (Reiteration) *Given a flag deduction with a line n with label A. Suppose that the last open subordinate proof, is a subordinate proof of the one at line n. Then we may extend it with a line m with the same label.*

n	A		
	$\vdots$		
		$\vdots$	
m		A	$R,\ n$

Definition A.1.5 ($\forall I$) *Let x be free for u in A. Given a flag deduction with last open subordinate proof beginning at line n with eigenvariable u. Suppose it includes a line i labelled A. Then we may extend the deduction with a new line k, labelled $\forall x.A[x/u]$ with motivation $\forall I, 3$, closing the subordinate proof.*

n	u			
		$\vdots$		
i		A		
		$\vdots$		
k	$\forall x.A[x/u]$		$\forall I,\ i$	

Definition A.1.6 ($\forall E$) *Given a flag deduction with a line i labelled $\forall x.A$, and a term t. Then we may extend the flag deduction with a line k labelled $A[t/x]$, under the proviso that i is in scope of k and t is free for x in A.*

i	$\forall x.A$	
	$\vdots$	
k	$A[t/x]$	$\forall E,\ i$

Definition A.1.7 (Identity-introduction) *Given a flag deduction and a term t. Then we may extend the flag deduction with a line i, labelled $t = t$, with motivation $= I$.*

$$\begin{array}{l|ll} i & t = t & =I \end{array}$$

A flag deduction may also start with an identity-introduction.

Definition A.1.8 (Identity-elimination) *Given a flag deduction with a line k with label $t = s$ and a line i with label B, we may extend it by creating a new line j with label $B[s//t]$, where $B[s//t]$ is the formula B in which zero or more occurrences of t are replaced by s.*

$$\begin{array}{l|ll} k & t = s & \\ & \vdots & \\ i & B & \\ & \vdots & \\ j & B[s//t] & =E_1,\ i,\ k \end{array} \qquad \begin{array}{l|ll} i & B & \\ & \vdots & \\ k & t = s & \\ & \vdots & \\ j & B[s//t] & =E_1,\ i,\ k \end{array}$$

Similarly, given a flag deduction with a line k with label $s = t$ and a line i with label B, we may extend it by creating a new line j with label $B[s//t]$:

$$\begin{array}{l|ll} k & s = t & \\ & \vdots & \\ i & B & \\ & \vdots & \\ j & B[s//t] & =E_2,\ i,\ k \end{array} \qquad \begin{array}{l|ll} i & B & \\ & \vdots & \\ k & s = t & \\ & \vdots & \\ j & B[s//t] & =E_2,\ i,\ k \end{array}$$

Where s is free for t in B.

After starting a subordinate proof by putting a variable u on line m, in general we will not keep the rest of the line empty. It is usual to put together two lines, by putting the first formula of the subordinate proof also on line m. See line 4 of Figure A.1.

Definition A.1.9 `FD` *is the system defined by the rules flag (Definition A.1.1), $\rightarrow I$ (Definition A.1.2), $\rightarrow E$ (Definition A.1.3), reiteration (Definition A.1.5), $\forall I$ (Definition A.1.6), $\forall E$ (Definition A.1.6), identity-introduction (Definition A.1.7) and identity-elimination (Definition A.1.8).*
The system $\texttt{FD}^-$ *is* `FD` *without identity-introduction (Definition A.1.7) and identity-elimination (Definition A.1.8).*

Definition A.1.10 *The* equivalence axioms *are:*

1. *$\forall x.x = x$ (identity);*

2. $\forall x.\forall y.(x = y \rightarrow y = x)$ *(symmetry);*

3. $\forall x.\forall y.\forall z.(x = y \rightarrow y = z \rightarrow x = z)$ *(transitivity).*

4. *For every n-ary relation symbol* R*:*
   $\forall x_1 \ldots \forall x_n.\forall y_1 \ldots \forall y_n.$
   $(x_1 = y_1 \rightarrow \ldots \rightarrow x_n = y_n \rightarrow \mathtt{R}(x_1, \ldots, x_n) \rightarrow \mathtt{R}(y_1, \ldots, y_n))$ *(relation axioms);*

5. *For every n-ary function symbol* F*:*
   $\forall x_1 \ldots \forall x_n.\forall y_1 \ldots \forall y_n.$
   $(x_1 = y_1 \rightarrow \ldots \rightarrow x_n = y_n \rightarrow \mathtt{F}(x_1, \ldots, x_n) = \mathtt{F}(y_1, \ldots, y_n))$ *(function axioms);*

Definition A.1.11 *The system* $\mathtt{FD_a}$ *is* $\mathtt{FD}^-$ *with the following rule added: Given a flag deduction, we may extend it with a new line that has one of the equivalence axioms as label. A flag deduction may also start with an equivalence axiom.*

A.2 Equivalence of FD and $\mathtt{FD_a}$

It can easily be verified that FD proves the equivalence axioms. We show now that the identity-introduction and the identity-elimination are derivable in $\mathtt{FD_a}$.

Lemma A.2.1 *The identity introduction is derivable in* $\mathtt{FD_a}$*.*

Proof

1	$\forall x.x = x$	identity
2	$t = t$	$\forall$E, 1

Lemma A.2.2 *The following rule is derivable in* $\mathtt{FD_a}$*:*

k	$t = s$
$\vdots$	$\vdots$
i	$q[t/x] = q[s/x]$

Proof We proof this by induction on the maximum number of nested function symbols, n, in q. If $n = 0$, q is a variable and we are finished (by reflexivity). Suppose $n = i + 1$ for some i, so $q = \mathtt{F}(p_1, \ldots, p_j)$. By induction, we can make

the following proof (here done for $j = 2$):

1	$t = s$	
$\vdots$	$\vdots$	(Ind.Hyp)
m	$p_1[t/x] = p_1[s/x]$	
$m+1$	$p_2[t/x] = p_2[s/x]$	
$m+2$	$\forall x_1.\forall x_2.\forall y_1.\forall y_2.$	
	$(x_1 = y_1 \rightarrow x_2 = y_2 \rightarrow \mathtt{F}(x_1, x_2) = \mathtt{F}(y_1, y_2))$	function axiom
$\vdots$	$\vdots$	($\forall$E steps)
$m+6$	$p_1[t/x] = p_1[s/x] \rightarrow p_2[t/x] = p_2[s/x] \rightarrow$	
	$\mathtt{F}(p_1[t/x], p_2[t/x]) = \mathtt{F}(p_1[s/x], p_2[s/x])$	
$m+7$	$p_2[t/x] = p_2[s/x] \rightarrow$	
	$\mathtt{F}(p_1[t/x], p_2[t/x]) = \mathtt{F}(p_1[s/x], p_2[s/x])$	$\Rightarrow$E, $m+6$, m
$m+8$	$\mathtt{F}(p_1[t/x], p_2[t/x]) = \mathtt{F}(p_1[s/x], p_2[s/x])$	$\Rightarrow$E, $m+7$, $m+1$

And because $\mathtt{F}(p_1[t/x], p_2[t/x]) \equiv (\mathtt{F}(p_1, p_2))[t/x]$ and $\mathtt{F}(p_1[s/x], p_2[s/x]) \equiv (\mathtt{F}(p_1, p_2))[s/x]$, we are done.

Lemma A.2.3 *The identity eliminations are derivable in* $\mathtt{FD}_\mathtt{a}$.

Proof

The proof is by induction on the number of connectives.

- Suppose B has no connectives. Remark that $B \equiv \mathtt{R}(p_1, \ldots, p_n)$ for some relation symbol R. Without loss of generality we may assume that t is not a real subterm of any of the ps (Lemma A.2.2). For the clarity of the exposition we will take R to be a 3-ary relation. We will treat the $=\mathrm{E}_1$ case that $t \neq p_1$, $t \neq p_2$, and $t = p_3$. Other cases are similar. So we start out with the following derivation ($k \neq i$):

k	$p_3 = s$
$\vdots$	$\vdots$
i	$\mathtt{R}(p_1, p_2, p_3)$

  First we take the appropriate relation axiom and we take p_m for both x_m

and y_m $(m = 1, 2)$. For x_3 we take p_3, but for y_3 we take s.

$i+1$	$\forall x_1.\forall x_2.\forall x_3.\forall y_1.\forall y_2.\forall y_3.$	
	$\quad (x_1 = y_1 \rightarrow x_2 = y_2 \rightarrow x_3 = y_3 \rightarrow$	
	$\quad\quad \mathtt{R}(x_1, x_2, x_3) \rightarrow \mathtt{R}(y_1, y_2, y_3))$	relation axiom
$\vdots$	$\vdots$	($\forall$E steps)
$i+7$	$p_1 = p_1 \rightarrow p_2 = p_2 \rightarrow p_3 = s \rightarrow$	
	$\quad \mathtt{R}(p_1, p_2, p_3) \rightarrow \mathtt{R}(p_1, p_2, s)$	$\forall$E, $i+1$

Then, we instantiate the identity for all $p_m \neq t$.

$i+8$	$\forall x.x = x$	identity
$i+9$	$p_1 = p_1$	$\forall$E, $i+8$
$i+10$	$p_2 = p_2$	$\forall$E, $i+8$

Finally, we use this to eliminate the implications in the instantiated relation axiom.

$i+11$	$p_2 = p_2 \rightarrow p_3 = s \rightarrow$	
	$\quad \mathtt{R}(p_1, p_2, p_3) \rightarrow \mathtt{R}(p_1, p_2, s)$	$\Rightarrow$E, $i+7$, $i+9$
$i+12$	$p_3 = s \rightarrow \mathtt{R}(p_1, p_2, p_3) \rightarrow \mathtt{R}(p_1, p_2, s)$	$\Rightarrow$E, $i+11$, $i+9$
$i+13$	$\mathtt{R}(p_1, p_2, p_3) \rightarrow \mathtt{R}(p_1, p_2, s)$	$\Rightarrow$E, $i+12$, $i+10$
$i+14$	$\mathtt{R}(p_1, p_2, s)$	$\Rightarrow$E, $i+13$, i

In this way, we can replace any number of ts by ss in B.

The proof of $=\mathrm{E}_2$ is similar, but uses additionally the axiom of symmetry.

- Suppose that the lemma holds for all formulas A with less than $m+1$ connectives. Suppose B has $m+1$ connectives. We show here the case $=\mathrm{E}_1$. The case $=\mathrm{E}_2$ is similar.

1. $B \equiv B_0 \rightarrow B_1$.

i	$t = s$		
$\vdots$	$\vdots$		
k	$B_0 \rightarrow B_1$		
$k+1$	$\quad B_0[s//t]$		
$k+2$	$\quad t = s$		R, i
$\vdots$	$\quad \vdots$		(Ind.Hyp.=E_2)
j	$\quad B_0$		
$j+1$	$\quad B_1$		$\Rightarrow$E, k, j
$\vdots$	$\quad \vdots$		(Ind.Hyp.=E_1)
l	$\quad B_1[s//t]$		
$l+1$	$B_0[s//t] \rightarrow B_1[s//t]$		$\Rightarrow$I, $k+1$, l

And because $B_0[s//t] \rightarrow B_1[s//t] \equiv (B_0 \rightarrow B_1)[s//t]$, we are done.

2. $B \equiv \forall x.B_0$.

i	$t = s$			
$\vdots$	$\vdots$			
k	$\forall x.B_0$			
$k+1$	u	$t = s$		R, i
$k+2$		$\forall x.B_0$		R, k
$k+3$		$B_0[u/x]$		$\forall$E, $k+2$
$\vdots$		$\vdots$		(Ind.Hyp.=E_1)
j		$B_0[u/x][s//t]$		
$j+1$	$\forall x.(B_0[s//t])$			$\forall$I, j

Samenvatting

Bewijstheorie valt grofweg in twee delen uiteen: structurele bewijstheorie en "bewijsbaarheidstheorie".

Structurele bewijstheorie houdt zich bezig met formele bewijzen vanuit een wiskundig oogpunt. Transformaties van een bewijs, zoals het verwijderen van irrelevante delen en het vereenvoudigen van de argumentatie, maken dit gebied interessant.

Bewijsbaarheidstheorie, anderzijds, beschouwt wiskunde vanuit een logisch perspectief. Een typische vraag is: is deze wiskundige stelling bewijsbaar in dit formele systeem?

Natuurlijk is de bovenstaande indeling niet zo duidelijk. Als we bijvoorbeeld weten welke geometrische vorm een bewijs kan aannemen, zou het wel eens gemakkelijker kunnen zijn een bewijs voor een bepaalde stelling te vinden. Deze verdeling beschrijft echter goed de onderwerpen van de twee delen van dit proefschrift: Deel I valt in de structurele bewijstheorie en Deel II in de (toegepaste) bewijsbaarheidstheorie.

In Deel I bestuderen we een fragment van natuurlijke deductie, waaraan we het begrip *hergebruik van deelresultaten* hebben toegevoegd. Het is heel gebruikelijk in de wiskunde om een eerder verkregen resultaat meermaals aan te wenden. Toch wordt dit in Gentzens systeem voor natuurlijke deductie niet in aanmerking genomen.

Afleidingen waarin deelresultaten hergebruikt worden, hebben over het algemeen een ingewikkeldere structuur dan afleidingen waarin dat niet voorkomt. Daarom moeten de transformaties van deze afleidingen ook wat subtieler zijn. Dit is eenvoudig in te zien: een deelresultaat kan onnodig algemeen zijn voor één plek waar het gebruikt wordt, tegelijkertijd is het misschien niet meteen te vereenvoudigen vanwege het gebruik op andere plaatsen.

Hoofdstuk 1 geeft een historisch overzicht van natuurlijke deductie en haar weergave. We schetsen de ontwikkeling van deze theorie. Verder voeren we een bezwaar aan met betrekking tot al bestaande formalismes die hergebruik toestaan. Door ons natuurlijke-deductieformalisme met hergebruik, de zogenaamde *deductiegraphen*, binnen deze ontwikkeling te plaatsen, rechtvaardigen we ons onderzoek.

Hoofdstuk 2 voert deductiegraphen voor minimale propositierekening formeel in. Hiervoor hebben we wat nodig uit uit de theorie van ordeningen en wat

graphentheorie. We laten zien hoe andere bekende natuurlijke-deductieformalismes in deductiegraphen kunnen worden ingebed.

Hoofdstuk 3 bestudeert de transformaties die deductiegraphen ondergaan tijdens de eliminatie van *snedes*.

Hoofdstuk 4 behandelt simpel getypeerde λ-calculus en een contextcalculus met "let-bindingen" en brengt deductiegraphen in verband met deze formalismes. Hieruit verkrijgen we *sterke normalisatie* voor deductiegraphen. Dat wil zeggen: elke deductiegraph kan in een eindig aantal transformatiestappen omgevormd worden tot een deductiegraph zonder snedes. Het blijkt dat de contextcalculus met let-bindingen de deductiegraphen volledig beschrijft.

Hoofdstuk 5 begint met een kort overzicht van de multiplicatieve exponentiële lineaire logica (MELL) en "proof-nets", de bijbehorende grafische weergave hiervan. Op het eerste gezicht lijken deductiegraphen op proof-nets. Door onze aandacht te verleggen naar *deductiegraphen met expliciet hergebruik*, maken we de overeenkomst precies. Deze analogie is niet a priori vanzelfsprekend, omdat MELL-proof-nets en deductiegraphen van verschillende ideeën afstammen.

Hoofdstuk 6 toont dan *confluentie* van deductiegraphen met expliciet hergebruik. Dat betekent dat de volgorde waarin de snedes geëlimineerd worden niet van invloed is op het uiteindelijke resultaat. We verkrijgen dit met brute kracht: het bewijs bestaat uit een uitputtende gevalsanalyse.

Hoofdstuk 7 voegt tenslotte universele kwantificatie toe aan deductiegraphen.

Deel II gaat over met de computer geformaliseerde wiskunde.

Computers zijn goede boekhouders. Door een wiskundig bewijs zo te maken dat een computer het kan staven, zou de juistheid van het resultaat dus zeer geloofwaardig kunnen worden. Aan de andere kant brengt het gebruik van computers voor de formalisatie van wiskunde ook problemen met zich mee.

Eén van die problemen is de weergave van de formele wiskunde. Een computer kan een bewijs alleen staven, als alle details, zowel van de definities als van de redeneringen, aanwezig zijn. Voor een persoon is een formeel bewijs daardoor vaak erg omstandig en dus moeilijk te begrijpen.

Hoofdstuk 8 pakt het probleem van de weergave aan de hand van een casus aan. We vervangen definities en stellingen van een interactief algebradictaat door hun formele tegenhangers. Het doel van dit document, onderwijs, benadrukt het belang van leesbaarheid en helderheid. We verkrijgen enige mate van leesbaarheid door de formele wiskunde een ingewikkelde transformatie te laten ondergaan.

Deze casus roert ook een paar aangrenzende gebieden aan: het gebruik van bibliotheken van formele wiskunde en het zetten van formele wiskunde op het wereldwijde web.

Curriculum Vitae

Born on March 27, 1980 in Wageningen, the Netherlands.

1992-1998	Stedelijk Gymnasium Arnhem.
1998-2003	Study Mathematics (cum laude) Katholieke Universiteit Nijmegen.
2003-2007	Junior Researcher Foundations, Institute for Computing and Information Sciences Radboud Universiteit Nijmegen.
2006	Visitor (January-March) Laboratoire Preuves, Programmes et Systèmes Université Paris VII-Denis Diderot, France Partially supported by a Frye Stipendium (2005).
From 2007	Research Associate Mathematics University of Canterbury, New Zealand.

Titles in the IPA Dissertation Series since 2002

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.* Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity.* Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing.* Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing.* Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining.* Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL.* Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand.* Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations.* Faculty of Natural Sciences,

Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability.* Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions.* Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments.* Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression.* Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages.* Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations.* Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics.* Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems.* Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication.* Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell.* Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model.* Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems.* Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality.* Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and*

Applications. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition*. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*.

Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06